
Brian M. Adams, Mohamed S. Ebeida, Michael S. Eldred, Gianluca Geraci, John D. Jakeman, Kathryn A. Maupin, Jason A. Monschke, J. Adam Stephens, Laura P. Swiler, Dena M. Vigil, Timothy M. Wildey
Optimization and Uncertainty Quantification Department

William J. Bohnhoff
Radiation Transport Department

Keith R. Dalbey
Software Simulation and Data Processing Department

John P. Eddy
System Readiness and Sustainment Technologies Department

Joseph R. Frye, Russell W. Hooper
Multiphysics Applications Department

Kenneth T. Hu
W76-0/W88 Stockpile Systems Engineering Department

Patricia D. Hough, Mohammad Khalil
Quantitative Modeling and Analysis Department

Elliott M. Ridgway
Data-driven and Neural Computing
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185

Ahmad Rushdi
Electrical and Computer Engineering
University of California
One Shields Avenue
Davis, CA 95616-5294
Abstract

The Dakota toolkit provides a flexible and extensible interface between simulation codes and iterative analysis methods. Dakota contains algorithms for optimization with gradient and nongradient-based methods; uncertainty quantification with sampling, reliability, and stochastic expansion methods; parameter estimation with nonlinear least squares methods; and sensitivity/variance analysis with design of experiments and parameter study methods. These capabilities may be used on their own or as components within advanced strategies such as surrogate-based optimization, mixed integer nonlinear programming, or optimization under uncertainty. By employing object-oriented design to implement abstractions of the key components required for iterative systems analyses, the Dakota toolkit provides a flexible and extensible problem-solving environment for design and performance analysis of computational models on high performance computers.

This report describes the Dakota class hierarchies. It is derived from annotation of the source code and provides detailed class documentation, including all member functions and attributes.
## Contents

1 Dakota Developers Manual .............................................. 13
   1.1 Introduction .......................................................... 13
   1.2 Overview of Dakota .................................................. 13
   1.3 Services ............................................................... 18
   1.4 Development Practices and Guidance .............................. 18
   1.5 Additional Resources ................................................ 19

2 Coding Style Guidelines and Conventions ............................ 21
   2.1 Introduction ........................................................... 21
   2.2 C++/c Style Guidelines .............................................. 21
   2.3 File Naming Conventions ............................................ 23
   2.4 Class Documentation Conventions .................................. 24
   2.5 CMake Style Guidelines ............................................ 24

3 Instructions for Modifying Dakota’s Input Specification .......... 27
   3.1 XML Input Specification ............................................. 27
   3.2 Rebuild Generated Files ............................................ 30
   3.3 Update Parser Source NIDRProblemDescDB.cpp ................. 30
   3.4 Update Corresponding Data Classes ................................ 31
   3.5 Update Database Source ProblemDescDB.cpp ..................... 32
   3.6 Use get_<data_type>() Functions ................................... 33
   3.7 Update the Documentation .......................................... 33

4 Understanding Iterator Flow ............................................. 35

5 Interfacing with Dakota as a Library .................................. 37
   5.1 Introduction ........................................................... 37
   5.2 Basic Dakota library instantiation ................................ 38
   5.3 Configuring Dakota operation ...................................... 39
   5.4 Creating a simulator plugin interface .............................. 42
   5.5 Retrieving data after a run ........................................ 45
   5.6 Linking against the Dakota library ................................ 45

6 Performing Function Evaluations ...................................... 47
   6.1 Synchronous function evaluations .................................. 47
   6.2 Asynchronous function evaluations ................................ 47
   6.3 Analyses within each function evaluation ......................... 48
7 Working with Variable Containers and Views
  7.1 Storage in Variables ................................................. 49
  7.2 Storage in SharedVariablesData .................................... 50
  7.3 Active and inactive views .......................................... 51

8 Namespace Index ............................................................ 53
  8.1 Namespace List ....................................................... 53

9 Hierarchical Index .......................................................... 55
  9.1 Class Hierarchy ....................................................... 55

10 Class Index .................................................................. 61
  10.1 Class List .............................................................. 61

11 File Index .................................................................. 73
  11.1 File List .............................................................. 73

12 Namespace Documentation ................................................. 75
  12.1 Dakota Namespace Reference ....................................... 75
  12.2 SIM Namespace Reference .......................................... 163

13 Class Documentation ....................................................... 165
  13.1 ActiveSet Class Reference ........................................... 165
  13.2 ActiveSubspaceModel Class Reference ............................. 167
  13.3 AdaptedBasisModel Class Reference ............................... 174
  13.4 Analyzer Class Reference ............................................ 179
  13.5 ApplicationInterface Class Reference .............................. 186
  13.6 Approximation Class Reference .................................... 202
  13.7 ApproximationInterface Class Reference ....................... 209
  13.8 APPSEvalMgr Class Reference ....................................... 216
  13.9 AppsTraits Class Reference ......................................... 218
  13.10 ApreproWriter Class Reference ..................................... 219
  13.11 BaseConstructor Struct Reference ................................. 220
  13.12 BootstrapSampler< Data > Class Template Reference .......... 220
  13.13 BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > > Class Template Reference ........................................... 221
  13.14 BootstrapSamplerBase< Data > Class Template Reference ...... 222
  13.15 BootstrapSamplerWithGS< Data, Getter, Setter > Class Template Reference ........................................... 224
  13.16 callback_data Struct Reference .................................... 225
  13.17 COLINApplication Class Reference ............................... 225
  13.18 COLINTraits Class Reference ...................................... 228
  13.19 CollabHybridMetaIterator Class Reference .................... 229
  13.20 CommandLineHandler Class Reference ............................ 231
  13.21 CommandShell Class Reference ................................... 232
  13.22 ConcurrentMetaIterator Class Reference ....................... 233
  13.23 CONMINTraits Class Reference ................................... 236
  13.24 ConsoleRedirector Class Reference ............................... 237
  13.25 Constraints Class Reference ....................................... 239
  13.26 DataEnvironment Class Reference ................................ 248
  13.27 DataEnvironmentRep Class Reference ............................ 249
CONTENTS

13.77 MatchesWC Struct Reference ................................................. 429
13.78 MatlabInterface Class Reference ........................................... 429
13.79 MetaIterator Class Reference ............................................... 431
13.80 Minimizer Class Reference ................................................. 432
13.81 MixedVarConstraints Class Reference ..................................... 440
13.82 MixedVariables Class Reference ........................................... 441
13.83 Model Class Reference ..................................................... 443
13.84 MPIManager Class Reference .............................................. 478
13.85 MPIPackBuffer Class Reference ............................................ 479
13.86 MPIUnpackBuffer Class Reference ........................................ 481
13.87 NCSUTraits Class Reference ............................................... 484
13.88 NestedModel Class Reference ............................................. 484
13.89 NIDRProblemDescDB Class Reference ..................................... 495
13.90 NL2Res Struct Reference .................................................. 500
13.91 NL2SOLLeastSqTraits Class Reference ................................... 501
13.92 NLPOQLP Traits Class Reference ........................................ 502
13.93 NLSSOLLeastSqTraits Class Reference ................................... 503
13.94 NoDBBaseConstructor Struct Reference ................................ 504
13.95 NomadTraits Class Reference ............................................. 504
13.96 NonD Class Reference ..................................................... 506
13.97 NonDAdaptImpSampling Class Reference .................................. 517
13.98 NonDAdaptiveSampling Class Reference ................................ 521
13.99 NonDBayesCalibration Class Reference .................................. 527
13.100 NonDCalibration Class Reference ....................................... 536
13.101 NonDCubature Class Reference .......................................... 537
13.102 NonDDREAMBayesCalibration Class Reference .......................... 540
13.103 NonDExpansion Class Reference ........................................ 544
13.104 NonDGlobalEvidence Class Reference ................................... 550
13.105 NonDGlobalInterval Class Reference ................................... 552
13.106 NonDGlobalReliability Class Reference ................................. 555
13.107 NonDGlobalSingleInterval Class Reference ............................. 558
13.108 NonDP采样Sampling Class Reference .................................. 559
13.109 NonDGPSA BayesCalibration Class Reference ........................... 562
13.110 NonDIntegration Class Reference ....................................... 566
13.111 NonDInterval Class Reference ........................................... 569
13.112 NonDLHSEvidence Class Reference ...................................... 572
13.113 NonDLHSInterval Class Reference ....................................... 573
13.114 NonDLHSSampling Class Reference ..................................... 574
13.115 NonDLHSSingleInterval Class Reference ................................ 578
13.116 NonDLocalEvidence Class Reference .................................... 579
13.117 NonDLocalInterval Class Reference ..................................... 580
13.118 NonDLocalReliability Class Reference .................................. 583
13.119 NonDLocalSingleInterval Class Reference ................................ 593
13.120 NonDMultilevelPolynomialChaos Class Reference ....................... 594
13.121 NonDMultilevelSampling Class Reference ............................... 597
13.122 NonDMultilevelStochCollocation Class Reference ...................... 606
13.123 NonDPoDFDarts Class Reference ....................................... 608
13.124 NonDPolynomialChaos Class Reference .................................. 611
13.125 NonDQuadrature Class Reference ....................................... 617
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.175</td>
<td>ResultsFileError Class Reference</td>
<td>775</td>
</tr>
<tr>
<td>13.176</td>
<td>ResultsID Class Reference</td>
<td>775</td>
</tr>
<tr>
<td>13.177</td>
<td>ResultsManager Class Reference</td>
<td>776</td>
</tr>
<tr>
<td>13.178</td>
<td>ResultsNames Class Reference</td>
<td>778</td>
</tr>
<tr>
<td>13.179</td>
<td>RichExtrapVerification Class Reference</td>
<td>779</td>
</tr>
<tr>
<td>13.180</td>
<td>ScalingModel Class Reference</td>
<td>782</td>
</tr>
<tr>
<td>13.181</td>
<td>ScalingOptions Class Reference</td>
<td>788</td>
</tr>
<tr>
<td>13.182</td>
<td>ScilabInterface Class Reference</td>
<td>789</td>
</tr>
<tr>
<td>13.183</td>
<td>SensAnalysisGlobal Class Reference</td>
<td>790</td>
</tr>
<tr>
<td>13.184</td>
<td>SeqHybridMetalterator Class Reference</td>
<td>793</td>
</tr>
<tr>
<td>13.185</td>
<td>SensAnalysisInterface Class Reference</td>
<td>797</td>
</tr>
<tr>
<td>13.186</td>
<td>TrackerHTTP::Server Struct Reference</td>
<td>798</td>
</tr>
<tr>
<td>13.187</td>
<td>ScalingModel Class Reference</td>
<td>782</td>
</tr>
<tr>
<td>13.188</td>
<td>SharedApproxData Class Reference</td>
<td>798</td>
</tr>
<tr>
<td>13.189</td>
<td>SharedPecosApproxData Class Reference</td>
<td>804</td>
</tr>
<tr>
<td>13.190</td>
<td>SharedResponseData Class Reference</td>
<td>806</td>
</tr>
<tr>
<td>13.191</td>
<td>SharedResponseDataRep Class Reference</td>
<td>809</td>
</tr>
<tr>
<td>13.192</td>
<td>SharedSurfpackApproxData Class Reference</td>
<td>811</td>
</tr>
<tr>
<td>13.193</td>
<td>SharedVariablesData Class Reference</td>
<td>812</td>
</tr>
<tr>
<td>13.194</td>
<td>SharedVariablesDataRep Class Reference</td>
<td>818</td>
</tr>
<tr>
<td>13.195</td>
<td>SimulationModel Class Reference</td>
<td>823</td>
</tr>
<tr>
<td>13.196</td>
<td>SimulationResponse Class Reference</td>
<td>826</td>
</tr>
<tr>
<td>13.197</td>
<td>SNLLBase Class Reference</td>
<td>827</td>
</tr>
<tr>
<td>13.198</td>
<td>SNLLLeastSqTraits Class Reference</td>
<td>829</td>
</tr>
<tr>
<td>13.199</td>
<td>SNLLTraits Class Reference</td>
<td>830</td>
</tr>
<tr>
<td>13.200</td>
<td>SOLBase Class Reference</td>
<td>831</td>
</tr>
<tr>
<td>13.201</td>
<td>SpawnApplicInterface Class Reference</td>
<td>833</td>
</tr>
<tr>
<td>13.202</td>
<td>TabularDataTruncated Class Reference</td>
<td>834</td>
</tr>
<tr>
<td>13.203</td>
<td>TabularReader Class Reference</td>
<td>838</td>
</tr>
<tr>
<td>13.204</td>
<td>SurrBasedGlobalMinimizer Class Reference</td>
<td>839</td>
</tr>
<tr>
<td>13.205</td>
<td>SurrBasedGlobalTraits Class Reference</td>
<td>840</td>
</tr>
<tr>
<td>13.206</td>
<td>SurrBasedLocalMinimizer Class Reference</td>
<td>847</td>
</tr>
<tr>
<td>13.207</td>
<td>SurrBasedMinimizer Class Reference</td>
<td>851</td>
</tr>
<tr>
<td>13.208</td>
<td>TANA3Approximation Class Reference</td>
<td>856</td>
</tr>
<tr>
<td>13.209</td>
<td>TabularDataTruncated Class Reference</td>
<td>859</td>
</tr>
<tr>
<td>13.210</td>
<td>TabularWriter Class Reference</td>
<td>860</td>
</tr>
<tr>
<td>13.211</td>
<td>TANA3Approximation Class Reference</td>
<td>860</td>
</tr>
<tr>
<td>13.212</td>
<td>TaylorApproximation Class Reference</td>
<td>861</td>
</tr>
<tr>
<td>13.213</td>
<td>TestDriverInterface Class Reference</td>
<td>863</td>
</tr>
<tr>
<td>13.214</td>
<td>TrackerHTTP::Server Struct Reference</td>
<td>864</td>
</tr>
<tr>
<td>13.215</td>
<td>TraitsBase Class Reference</td>
<td>871</td>
</tr>
<tr>
<td>13.216</td>
<td>UsageTracker Class Reference</td>
<td>873</td>
</tr>
<tr>
<td>13.217</td>
<td>VLight Struct Reference</td>
<td>875</td>
</tr>
<tr>
<td>13.218</td>
<td>VReal Struct Reference</td>
<td>876</td>
</tr>
<tr>
<td>13.219</td>
<td>Variables Class Reference</td>
<td>876</td>
</tr>
<tr>
<td>13.220</td>
<td>Verification Class Reference</td>
<td>877</td>
</tr>
<tr>
<td>13.221</td>
<td>VLight Struct Reference</td>
<td>887</td>
</tr>
<tr>
<td>13.222</td>
<td>VReal Struct Reference</td>
<td>888</td>
</tr>
<tr>
<td>13.223</td>
<td>VStr Struct Reference</td>
<td>889</td>
</tr>
</tbody>
</table>
Chapter 1

Dakota Developers Manual

Author

1.1 Introduction
The Dakota software (http://dakota.sandia.gov/) delivers advanced parametric analysis techniques enabling quantification of margins and uncertainty, risk analysis, model calibration, and design exploration with computational models. Dakota contains algorithms for optimization with gradient and nongradient-based methods, uncertainty quantification with sampling, reliability, stochastic expansion, and interval estimation methods, parameter estimation with nonlinear least squares methods, and sensitivity/variance analysis with design of experiments and parameter study capabilities. (Solution verification and Bayesian approaches are also in development.) These capabilities may be used on their own or as components within advanced algorithms such as surrogate-based optimization, mixed integer nonlinear programming, mixed aleatory-epistemic uncertainty quantification, or optimization under uncertainty. By employing object-oriented design to implement abstractions of the key components required for iterative systems analyses, the Dakota toolkit provides a flexible problem-solving environment for design and performance analysis of computational models on high performance computers.

The Developers Manual focuses on documentation of Dakota design principles and class structures; it derives principally from annotated source code. For information on input command syntax, refer to the Reference Manual[1], and for more details on Dakota features and capabilities, refer to the Users Manual.

1.2 Overview of Dakota
In Dakota, the environment manages execution modes and input/output streams and defines the top-level iterator. This top-level iterator may be either a standard iterator or a meta-iterator. In the former case, the iterator identifies a model and the environment executes the iterator on the model to perform a single study. In the latter case, iterator recursions are present and sub-iterations may identify their own models. In both cases, models may contain additional recursions in the case of nested iteration or surrogate modeling. In a simple example, a hybrid meta-iterator might manage a global optimizer operating on a low-fidelity model that feeds promising design points into a local optimizer operating on a high-fidelity model. And in a more advanced example, a surrogate-based optimization under uncertainty approach would employ an uncertainty quantification iterator nested within an
optimization iterator and would employ truth models contained within surrogate models. Thus, iterators and models provide both stand-alone capabilities as well as building blocks for more sophisticated studies.

A model contains a set of variables, an interface, and a set of responses, and the iterator operates on the model to map the variables into responses using the interface. Each of these components is a flexible abstraction with a variety of specializations for supporting different types of iterative studies. In a Dakota input file, the user specifies these components through environment, method, model, variables, interface, and responses keyword specifications.

The use of class hierarchies provides a mechanism for extensibility in Dakota components. In each of the various class hierarchies, adding a new capability typically involves deriving a new class and providing a set of virtual function redefinitions. These redefinitions define the coding portions specific to the new derived class, with the common portions already defined at the base class. Thus, with a small amount of new code, the existing facilities can be extended, reused, and leveraged for new purposes. The following sections tour Dakota’s class organization.

1.2.1 Environment

Class hierarchy: Environment.
Environments provide the top level abstraction for managing different execution modes and managing input and output streams. Specific environments include:

- ExecutableEnvironment: the environment for execution of Dakota as a stand-alone application.
- LibraryEnvironment: the environment for execution of Dakota as an embedded library service.

1.2.2 Iterators

Class hierarchy: Iterator. Iterator implementations may choose to split operations up into run-time phases as described in Understanding Iterator Flow.

The iterator hierarchy contains a variety of iterative algorithms for optimization, uncertainty quantification, nonlinear least squares, design of experiments, and parameter studies. The hierarchy is divided into MetaIterator, Minimizer, and Analyzer branches.

The MetaIterator classes manage sequencing and collaboration among multiple methods with support for concurrent iterator parallelism. Methods include:

- SeqHybridMetaIterator: hybrid minimization using a set of iterators employing a corresponding set of models of varying fidelity. The sequential hybrid passes the best solutions from one method in as the starting points of the next method in the sequence.
- CollabHybridMetaIterator: hybrid minimization employing collaboration and sharing of response data among methods during the course if iteration. This class is currently a placeholder.
- EmbedHybridMetaIterator: hybrid minimization involving periodic use of a local search method for refinement during the iteration of an outer global method. This class is currently a placeholder.
- ConcurrentMetaIterator: two similar algorithms are available: (1) multi-start iteration from several different starting points, and (2) pareto set optimization for several different multi-objective weightings. Employs a single iterator with a single model, but runs multiple instances of the iterator concurrently for different settings within the model.

The Minimizer classes address optimization and deterministic calibration and are grouped into:
1.2. OVERVIEW OF DAKOTA

- **Optimization**: Optimizer provides a base class for gradient-based (e.g., CONMINOptimizer and SNL-LOptimizer) and derivative-free (e.g., NCSUOptimizer, JEGAOptimizer) optimization solvers. Most of these are wrappers for third-party libraries that implement the optimization algorithms. Classes APPSEvalMgr and COLINApplication provide the function evaluation interface for APPSOptimizer and COLINOptimizer, respectively.

- **Parameter estimation**: LeastSq provides a base class for NL2SOLLeastSq, a least-squares solver based on NL2SOL, SNLLLeastSq, a Gauss-Newton least-squares solver, and NLSSOLLeastSq, an SQP-based least-squares solver.

- **Surrogate-based minimization (both optimization and nonlinear least squares)**: SurrBasedMinimizer provides a base class for SurrBasedLocalMinimizer, SurrBasedGlobalMinimizer, and EffGlobalMinimizer. The surrogate-based local and global methods employ a single iterator with any of the available Surrogate-Model capabilities (local, multipoint, or global data fits or hierarchical approximations) and perform a sequence of approximate optimizations, each involving build, optimize, and verify steps. The efficient global method, on the other hand, hard-wires a recursion involving Gaussian process surrogate models coupled with the DIRECT global optimizer to maximize an expected improvement function.

The Analyzer classes are grouped into:

- **Uncertainty quantification**: NonD provides a base class for non-deterministic methods in several categories:
  - Sampling: NonDSampling is further specialized with the NonDLHSSampling class for Latin hyper-cube and Monte Carlo sampling, and a number of other classes supporting incremental and adaptive sampling such as NonDAadaptImpSampling for multi-modal adaptive importance sampling.
  - Reliability Analysis: NonDReliability is further specialized with local and global methods (NonDLocalReliability and NonDGlobalReliability). NonDPOFDarts implements a computational geometry-based reliability method.
  - Stochastic Expansions: NonDExpansion includes specializations for generalized polynomial chaos (NonDPolynomialChaos) and stochastic collocation (NonDStochCollocation) and is supported by the NonDIntegration helper class, which supplies cubature, tensor-product quadrature and Smolyak sparse grid methods (NonDCubature, NonDQuadrature, and NonDSparseGrid).
  - Bayesian Calibration: NonDCalibration provides a base class for nondeterministic calibration methods with specialization to Bayesian calibration in NonDBayesCalibration, and specific implementations such as NonDQUESOBayesCalibration.
  - NonDInterval provides a base class for epistemic interval-based UQ methods. Three interval analysis approaches are provided: LHS (NonDLHSSInterval), efficient global optimization (NonDGlobalInterval), and local optimization (NonDLocalInterval). Each of these three has specializations for single interval and Dempster-Shafer Theory of Evidence approaches.

- **Parameter studies and design of experiments**: PStudyDACE provides a base class for ParamStudy, which provides capabilities for directed parameter space interrogation, PSUADEDesignCompExp, which provides access to the Morris One-At-a-Time (MOAT) method for parameter screening, and DDACEDesignCompExp and FSUDesignCompExp, which provide for parameter space exploration through design and analysis of computer experiments. NonDLHSSampling from the uncertainty quantification branch also supports design of experiments when in active all variables mode.

- **Solution verification studies**: Verification provides a base class for RichExtrpVerification (verification via Richardson extrapolation) and other solution verification methods in development.
1.2.3 Models

Class hierarchy: Model.

The model classes are responsible for mapping variables into responses when an iterator makes a function evaluation request. There are several types of models, some supporting sub-iterators and sub-models for enabling layered and nested relationships. When sub-models are used, they may be of arbitrary type so that a variety of recursions are supported.

- SimulationModel: variables are mapped into responses using a simulation-based Interface object. No sub-iterators or sub-models are used.

- SurrogateModel: variables are mapped into responses using an approximation. The approximation is built and/or corrected using data from a sub-model (the truth model) and the data may be obtained using a sub-iterator (a design of experiments iterator). SurrogateModel has two derived classes: DataFitSurrModel for data fit surrogates and HierarchSurrModel for hierarchical models of varying fidelity. The relationship of the sub-iterators and sub-models is considered to be "layered" since they are not used as part of every response evaluation on the top level model, but rather used periodically in surrogate update and verification steps.

- NestedModel: variables are mapped into responses using a combination of an optional Interface and a sub-iterator/sub-model pair. The relationship of the sub-iterators and sub-models is considered to be "nested" since they are used to perform a complete iterative study as part of every response evaluation on the top level model.

- RecastModel: recasts the inputs and outputs of a sub-model for the purposes of variable transformations (e.g., variable scaling, transformations to standardized random variables) and problem reformulation (e.g., multi-objective optimization, response scaling, augmented Lagrangian merit functions, expected improvement).

1.2.4 Variables

Class hierarchy: Variables.

The Variables class hierarchy manages design, aleatory uncertain, epistemic uncertain, and state variable types for continuous, discrete integer, and discrete real domain types. This hierarchy is specialized according to how the domain types are managed:

- MixedVariables: domain type distinctions are retained, such that separate continuous, discrete integer, and discrete real domain types are managed. This is the default Variable perspective, and draws its name from "mixed continuous-discrete" optimization.

- RelaxedVariables: domain types are combined through relaxation of discrete constraints; i.e., continuous and discrete variables are merged into continuous arrays through relaxation of integrality (for discrete integer ranges) or set membership (for discrete integer or discrete real sets) requirements. The branch and bound minimizer is the only method using this approach at present.

Whereas domain types are defined based on the derived Variables class selection, the selection of active variable types is handled within each of these derived classes using variable views. These permit different algorithms to work on different subsets of variables. Data shared among Variables instances is stored in SharedVariablesData. For details on managing variables, see Working with Variable Containers and Views.

The Constraints hierarchy manages bound, linear, and nonlinear constraints and utilizes the same specializations for managing bounds on the variables (see MixedVarConstraints and RelaxedVarConstraints).
1.2. OVERVIEW OF DAKOTA

1.2.5 Interfaces

Class hierarchy: Interface.

Interfaces provide access to simulation codes or, conversely, approximations based on simulation code data. In the simulation case, an ApplicationInterface is used. ApplicationInterface is specialized according to the simulation invocation mechanism, for which the following nonintrusive approaches are supported:

- **SysCallApplicInterface**: the simulation is invoked using a system call (the C function `system()`). Asynchronous invocation utilizes a background system call. Utilizes the CommandShell utility.

- **ForkApplicInterface**: the simulation is invoked using a fork (the `fork/exec/wait` family of functions). Asynchronous invocation utilizes a nonblocking fork.

- **SpawnApplicInterface**: for Windows, fork is replaced by spawn. Asynchronous invocation utilizes a nonblocking spawn.

Fork and Spawn are inherited from ProcessHandleApplicInterface and System and ProcessHandle are inherited from ProcessApplicInterface. A semi-intrusive approach is also supported by:

- **DirectApplicInterface**: the simulation is linked into the Dakota executable and is invoked using a procedure call. Asynchronous invocations will utilize nonblocking threads (capability not yet available). Specializations of the direct interface are implemented in MatlabInterface, PythonInterface, ScilabInterface, and (for built-in testers) TextDriverInterface, while examples of plugin interfaces for library mode in serial and parallel, respectively, are included in SerialDirectApplicInterface and ParallelDirectApplicInterface.

Scheduling of jobs for asynchronous local, message passing, and hybrid parallelism approaches is performed in the ApplicationInterface class, with job initiation and job capture specifics implemented in the derived classes.

In the approximation case, global, multipoint, or local data fit approximations to simulation code response data can be built and used as surrogates for the actual, expensive simulation. The interface class providing this capability is

- **ApproximationInterface**: builds an approximation using data from a truth model and then employs the approximation for mapping variables to responses. This class contains an array of Approximation objects, one per response function, which support a variety of approximation types using the different Approximation derived classes. These include SurfpackApproximation (provides kriging, MARS, moving least squares, neural network, polynomial regression, and radial basis functions), GaussProcApproximation (Gaussian process models), PecosApproximation (multivariate orthogonal and Lagrange interpolation polynomials from Pecos), TANA3Approximation (two-point adaptive nonlinearity approximation), and TaylorApproximation (local Taylor series).

which is an essential component within the DataFitSurrModel capability described above in Models.

1.2.6 Responses

Class: Response.

The Response class provides an abstract data representation of response functions and their first and second derivatives (gradient vectors and Hessian matrices). These response functions can be interpreted as objective functions and constraints (optimization data set), residual functions and constraints (least squares data set), or generic response functions (uncertainty quantification data set). This class is not currently part of a class hierarchy, since the abstraction has been sufficiently general and has not required specialization.
1.3 Services

A variety of services and utilities are used in Dakota for parallel computing, failure capturing, restart, graphics, etc. An overview of the classes and member functions involved in performing these services is included here.

- **Multilevel parallel computing:** Dakota supports multiple levels of nested parallelism. A meta-iterator can manage concurrent iterators, each of which manages concurrent function evaluations, each of which manages concurrent analyses executing on multiple processors. Partitioning of these levels with MPI communicators is managed in ParallelLibrary and scheduling routines for the levels are part of IteratorScheduler, ApplicationInterface, and ForkApplicationInterface.

- **Option management:** Global options controlling behavior are managed in ProgramOptions, with the help of command-line option parsing in CommandLineHandler.

- **Parsing:** Dakota employs NIDR (New Input Deck Reader) via Dakota::ProblemDescDB::parse_inputs to parse user input files. NIDR uses the keyword handlers in the NIDRProblemDescDB derived class to populate data within the ProblemDescDB base class, which maintains a DataEnvironment specification and lists of DataMethod, DataModel, DataVariables, DataInterface, and DataResponses specifications. Procedures for modifying the parsing subsystem are described in Instructions for Modifying Dakota’s Input Specification.

- **Failure capturing:** Simulation failures can be trapped and managed using exception handling in ApplicationInterface and its derived classes.

- **Restart:** Dakota maintains a record of all function evaluations both in memory (for capturing any duplication) and on the file system (for restarting runs). Restart options are managed through ProgramOptions (with the help of CommandLineHandler); file management in OutputManager; and restart file insertions occur in ApplicationInterface. The dakota_restart_util executable, built from restart_util.cpp, provides a variety of services for interrogating, converting, repairing, concatenating, and post-processing restart files.

- **Memory management:** Dakota employs the techniques of reference counting and representation sharing through the use of letter-envelope and handle-body idioms (Coplien, ”Advanced C++”). The former idiom provides for memory efficiency and enhanced polymorphism in the following class hierarchies: Environment, Iterator, Model, Variables, Constraints, Interface, ProblemDescDB, and Approximation. The latter idiom provides for memory efficiency in data-intensive classes which do not involve a class hierarchy. The Response and parser data (DataEnvironment, DataMethod, DataModel, DataVariables, DataInterface, and DataResponses) classes use this idiom. When managing reference-counted data containers (e.g., Variables or Response objects), it is important to properly manage shallow and deep copies, to allow for both efficiency and data independence as needed in a particular context.

- **Graphics and Output:** Dakota provides 2D iteration history graphics using Motif widgets. Graphics data can also be cataloged in a tabular data file for post-processing with 3rd party tools such as Matlab, Tecplot, etc. These capabilities are encapsulated within the Graphics class. An experimental results database is implemented in ResultsManager and ResultsDBAny. Options for controlling output and facilities for managing it are in OutputManager.

1.4 Development Practices and Guidance

The following links provide guidance for core software components or specific development activities:

- **Coding Style Guidelines and Conventions** - coding practices used by the Dakota development team.
1.5. ADDITIONAL RESOURCES

- **Instructions for Modifying Dakota’s Input Specification** - how to interact with NIDR and the associated Dakota classes.
- **Interfacing with Dakota as a Library** - embed Dakota as a service within your application.
- **Understanding Iterator Flow** - explanation of the full granularity of steps in Iterator execution.
- **Performing Function Evaluations** - an overview of the classes and member functions involved in performing function evaluations synchronously or asynchronously.
- **Working with Variable Containers and Views** - discussion of data storage for variables and explanation of active and inactive views of this data.

1.5 Additional Resources

Additional development resources include:

- The Dakota Developer Portal linked from [http://dakota.sandia.gov/content/developer-portal/](http://dakota.sandia.gov/content/developer-portal/) includes information on getting started as a developer and links to project management resources.
- Project web pages are maintained at [http://dakota.sandia.gov/](http://dakota.sandia.gov/) including links to frequently asked questions, documentation, publications, mailing lists, and other resources.
Chapter 2

Coding Style Guidelines and Conventions

2.1 Introduction

Common code development practices can be extremely useful in multiple developer environments. Particular styles for code components lead to improved readability of the code and can provide important visual cues to other developers. Much of this recommended practices document is borrowed from the CUBIT mesh generation project, which in turn borrows its recommended practices from other projects, yielding some consistency across Sandia projects. While not strict requirements, these guidelines suggest a best-practices starting point for coding in Dakota.

2.2 C++/c Style Guidelines

Style guidelines involve the ability to discern at a glance the type and scope of a variable or function.

2.2.1 Class and variable styles

Class names should be composed of two or more descriptive words, with the first character of each word capitalized, e.g.:

```cpp
class ClassName;
```

Class member variables should be composed of two or more descriptive words, with the first character of the second and succeeding words capitalized, e.g.:

```cpp
double classMemberVariable;
```

Temporary (i.e. local) variables are lower case, with underscores separating words in a multiple word temporary variable, e.g.:

```cpp
int temporary_variable;
```

Constants (i.e. parameters) and enumeration values are upper case, with underscores separating words, e.g.:

```cpp
const double CONSTANT_VALUE;
```
2.2.2 Function styles

Function names are lower case, with underscores separating words, e.g.:

```c
int function_name();
```

There is no need to distinguish between member and non-member functions by style, as this distinction is usually clear by context. This style convention allows member function names which set and return the value of a similarly-named private member variable, e.g.:

```c
int memberVariable;
void member_variable(int a) { // set
    memberVariable = a;
}
int member_variable() const { // get
    return memberVariable;
}
```

In cases where the data to be set or returned is more than a few bytes, it is highly desirable to employ const references to avoid unnecessary copying, e.g.:

```c
void continuous_variables(const RealVector& c_vars) { // set
    continuousVariables = c_vars;
}
const RealVector& continuous_variables() const { // get
    return continuousVariables;
}
```

Note that it is not necessary to always accept the returned data as a const reference. If it is desired to be able change this data, then accepting the result as a new variable will generate a copy, e.g.:

```c
// reference to continuousVariables cannot be changed
const RealVector& c_vars = model.continuous_variables();
// local copy of continuousVariables can be changed
RealVector c_vars = model.continuous_variables();
```

2.2.3 Miscellaneous

Appearance of typedefs to redefine or alias basic types is isolated to a few header files (`data_types.h`, `template_defs.h`), so that issues like program precision can be changed by changing a few lines of typedefs rather than many lines of code, e.g.:

```c
typedef double Real;
```

xemacs is the preferred source code editor, as it has C++ modes for enhancing readability through color (turn on “Syntax highlighting”). Other helpful features include “Paren highlighting” for matching parentheses and the “New Frame” utility to have more than one window operating on the same set of files (note that this is still the same edit session, so all windows are synchronized with each other). Window width should be set to 80 internal columns, which can be accomplished by manual resizing, or preferably, using the following alias in your shell resource file (e.g., .cshrc):

```bash
alias xemacs "xemacs -g 81x63"
```

where an external width of 81 gives 80 columns internal to the window and the desired height of the window will vary depending on monitor size. This window width imposes a coding standard since you should avoid line wrapping by continuing anything over 80 columns onto the next line.

Indenting increments are 2 spaces per indent and comments are aligned with the code they describe, e.g.:
2.3. FILE NAMING CONVENTIONS

void abort_handler(int code)
{
    int initialized = 0;
    MPI_Init(&initialized);
    if (initialized) {
        // comment aligned to block it describes
        int size;
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if (size>1)
            MPI_Abort(MPI_COMM_WORLD, code);
        else
            exit(code);
    }
    else
        exit(code);
}

Also, the continuation of a long command is indented 2 spaces, e.g.:

const String& iterator_scheduling
    = problem_db.get_string("strategy.iterator_scheduling");

and similar lines are aligned for readability, e.g.:

cout << "Numerical gradients using " << finiteDiffStepSize*100. << "\%
    
    << finiteDiffType << " differences\nto be calculated by the "
    
    << methodSource << " finite difference routine." << endl;

Lastly, #ifdef’s are not indented (to make use of syntax highlighting in xemacs).

2.3 File Naming Conventions

In addition to the style outlined above, the following file naming conventions have been established for the Dakota project.

File names for C++ classes should, in general, use the same name as the class defined by the file. Exceptions include:

- with the introduction of the Dakota namespace, base classes which previously utilized prepended Dakota identifiers can now safely omit the identifiers. However, since file names do not have namespace protection from name collisions, they retain the prepended Dakota identifier. For example, a class previously named DakotaModel which resided in DakotaModel.cpp/hpp, is now Dakota::Model (class Model in namespace Dakota) residing in the same filenames. The retention of the previous filenames reduces the possibility of multiple instances of a Model.hpp causing problems. Derived classes (e.g., NestedModel) do not require a prepended Dakota identifier for either the class or file names.

- in a few cases, it is convenient to maintain several closely related classes in a single file, in which case the file name may reflect the top level class or some generalization of the set of classes (e.g., DakotaResponse.[CH] files contain Dakota::Response and Dakota::ResponseRep classes, and DakotaBinStream.[CH] files contain the Dakota::BiStream and Dakota::BoStream classes).

The type of file is determined by one of the four file name extensions listed below:

- .hpp A class header file ends in the suffix .hpp. The header file provides the class declaration. This file does not contain code for implementing the methods, except for the case of inline functions. Inline functions are to be placed at the bottom of the file with the keyword inline preceding the function name.

- .cpp A class implementation file ends in the suffix .cpp. An implementation file contains the definitions of the members of the class.
• .h A header file ends in the suffix .h. The header file contains information usually associated with procedures. Defined constants, data structures and function prototypes are typical elements of this file.

• .c A procedure file ends in the suffix .c. The procedure file contains the actual procedures.

2.4 Class Documentation Conventions

Class documentation uses the doxygen tool available from http://www.doxygen.org and employs the J-AVA-doc comment style. Brief comments appear in header files next to the attribute or function declaration. Detailed descriptions for functions should appear alongside their implementations (i.e., in the .cpp files for non-inlined, or in the headers next to the function definition for inlined). Detailed comments for a class or a class attribute must go in the header file as this is the only option.

NOTE: Previous class documentation utilities (class2frame and class2html) used the ”//” comment style and comment blocks such as this:

```c++
// Class: Model
// Description: The model to be iterated by the Iterator.
// Contains Variables, Interface, and Response objects.
// Owner: Mike Eldred
```

These tools are no longer used, so remaining comment blocks of this type are informational only and will not appear in the documentation generated by doxygen.

2.5 CMake Style Guidelines

Dakota conventions for CMake files, such as CMakeLists.txt, FooConfig.cmake, etc., follow. Our goal is ease of reading, maintenance, and support, similar to the C++ code itself. Current CMake versions and build hints are maintained at the Developer Portal http://dakota.sandia.gov/developer/.

2.5.1 CMake Code Formatting

• Indentation is 2 spaces, consistent with Dakota C++ style.
• Lines should be kept to less than 80 chars per line where possible.
• Wrapped lines may be indented two spaces or aligned with prior lines.
• For ease of viewing and correctness checking in Emacs, a customization file is available: http://www.-cmake.org/CMakeDocs/cmake-mode.el

2.5.2 CMake Variable Naming Conventions

These variable naming conventions are especially important for those that ultimately become preprocessor defines and affect compilation of source files.

• Classic/core elements of the CMake language are set in lower_case, e.g., option, set, if, find_library.
• Static arguments to CMake functions and macros are set in UPPER_CASE, e.g. REQUIRED, NO_MODULE, QUIET.
• Minimize ”global” variables, i.e., don’t use 2 variables with the same meaning when one will do the job.
2.5. CMAKE STYLE GUIDELINES

- Feature toggling: when possible, use the "HAVE_<pkg/feature>" convention already in use by many CMake-enabledTPLs, e.g.,

```bash
$ grep HAVE_SYSTEM Dakota/src/CMakeLists.txt

check_function_exists(system HAVE_SYSTEM)
if(HAVE_SYSTEM)
  add_definitions("-DHAVE_SYSTEM")
endif(HAVE_SYSTEM)

$ grep HAVE_CONMIN Dakota/src/CMakeLists.txt Dakota/packages/CMakeLists.txt
Dakota/src/CMakeLists.txt:if(HAVE_CONMIN)
Dakota/src/CMakeLists.txt:endif(HAVE_CONMIN)
Dakota/packages/CMakeLists.txt:option(HAVE_CONMIN "Build the CONMIN package." ON)
Dakota/packages/CMakeLists.txt:if(HAVE_CONMIN)
Dakota/packages/CMakeLists.txt:endif(HAVE_CONMIN)
```

- When a variable/preprocessor macro could result in name clashes beyond Dakota scope, e.g., for library-mode users, consider prefixing the "HAVE_<pkg>" name with DAKOTA, e.g. DAKOTA_HAVE_MPI. Currently, MPI is the only use case for such a variable in Dakota, but many examples can be found in the CMake Modules source, e.g.

```bash
grep _HAVE_<cmake_prefix_dir>/share/cmake-2.8/Modules/*
```
Chapter 3

Instructions for Modifying Dakota’s Input Specification

To modify Dakota’s input specification (for maintenance or addition of new input syntax), specification maintenance mode must be enabled at Dakota configure time with the `-DENABLE_SPEC_MAINT` option, e.g.,

```
./cmake -DENABLE_SPEC_MAINT:BOOL=ON ..
```

This will enable regeneration of NIDR and Dakota components which must be updated following a spec change.

3.1 XML Input Specification

The authoritative source for valid Dakota input grammar is `dakota/src/dakota.xml`. The schema defining valid content for this XML file is in `dakota/src/dakota.xsd`. NIDR remains Dakota’s user input file parser, so `dakota.xml` is translated to `dakota/src/dakota.input.nspec` during the Dakota build process. To update the XML input definition:

- Make sure `ENABLE_SPEC_MAINT` is enabled in your build and necessary Java development tools are installed (see below).
- Edit the XML spec in `dakota.xml`.
- Perform a make in `dakota.build/src` which will regenerate `dakota.source/src/dakota.input.nspec` and related file.
- Review that any changes induced in the `dakota.input.nspec` file are as expected.
- Proceed with verifying code changes and making downstream parse handler changes as normal (described below).
- Commit the modified `dakota.xml`, `dakota.input.nspec`, and other files generated to `dakota.source/src` along with your other code changes.

3.1.1 XML Build Requirements

Editing the XML and then compiling Dakota requires
• Java Development Kit (JDK) providing the Java compiler javac. Java 6 (version 1.6) or newer should work, with Java 8 recommended. Can satisfy on RHEL6 with RPM packages java-1.8.0-openjdk-devel and java-1.8.0-openjdk. This is needed to build the Java-based XML to NIDR translator. If this becomes too burdensome, we can check in the generated xml2nidr.jar file.

3.1.2 XML Editing Tools

The following tools will make editing dakota.input.xml easier.

• **Recommended: Eclipse Web Tools Platform.** Includes both graphical and text editors.

  1. Download Eclipse Standard (Classic)
  2. Configure proxy if needed, setting to manual: Window > Preferences > General > Network Connection > Proxy
  3. Install Web Tools Platform
     - Help > Install New Software
     - Work With: Kepler - [http://download.eclipse.org/releases/kepler](http://download.eclipse.org/releases/kepler)
     - Search ”Eclipse X” and install two packages under Web, XML, Java
       * Eclipse XML Editors and Tools
       * Eclipse XSL Developer Tools
     - Optionally install C/C++ Development Tools
  4. Optional: add Subclipse for subversion (Subversive is the other major competing tool and I don’t think requires JavaHL) Help > Install New Software Work With: [http://subclipse.tigris.org/update_1.6.x](http://subclipse.tigris.org/update_1.6.x) Install Subclipse On Linux: yum install subversion-javahl.x86_64
  5. Alternately install Eclipse for Java or Eclipse Java EE development which includes webtools, then optionally add subclipse and C/C++ dev

• **Alternate: Emacs or your usual editor.** For example, Emacs supports an Nxml mode. You can tell it where to find the schema, edit XML, and have it perform validation against the schema. See help at [http://www.gnu.org/software/emacs/manual/html_mono/nxml-mode.html](http://www.gnu.org/software/emacs/manual/html_mono/nxml-mode.html)

• **Other Suggested Alternates:** XMLSpy, DreamWeaver, XML Copy Editor

3.1.3 XML Features (with map to NIDR)

Out of necessity, Dakota XML dakota.xml closely mirrors dakota.input.nspec. Valid Dakota input grammar is constrained by dakota.xml, an XML document which must validate against dakota.xsd. The top-level element of interest is `<input>`, which is comprised of a sequence of content elements (keywords, alternates, etc.), which may themselves contain additional child content elements. The key content types are:

• **Keyword** ( `<keyword>`)**: specified with the `<keyword>` element whose definition is given by keyword-Type in dakota.xsd. The required attributes are:

  - **name**: The keyword name (lower case with underscores) as it will be given in user input; must follow same uniqueness rules are historical NIDR. User input is allowed in mixed case, but the XML must use lower case names.

  Since the NIDR parser allows keyword abbreviation, you **must** not add a keyword that could be mis-interpreted as an abbreviation for a different keyword within the same top-level keyword, such as
"environment" and "method". For example, adding the keyword "expansion" within the method specification would be a mistake if the keyword "expansion_factor" already was being used in this block. The NIDR input is somewhat order-dependent, allowing the same keyword to be reused multiple times in the specification. This often happens with aliases, such as lower_bounds, upper_bounds and initial_point. Ambiguities are resolved by attaching a keyword to the most recently seen context in which it could appear, if such exists, or to the first relevant context that subsequently comes along in the input file.

- **code:** The verbatim NIDR handler to be invoked when this keyword parsed. In NIDR this was specified with \{N_macro(...)\}.

Optional/useful parser-related elements/attributes in order of importance are:

- **param sub-element:** Parameters and data types: A keyword may have an associated parameter element with a specified data type: \(<param type="PARAMTYPE" />\). NIDR data types remain the same (INTEGER, REAL, STRING and LISTS thereof, but new data types INPUT_FILE and OUTPUT_FILE add convenience for the GUI, mapping to STRING for NIDR purposes. Parameters can also include attributes constraint, in_taglist, or taglist, which are used to help validate the user-specified parameter value. For example constraint \(=> 0\) LEN normal uncertain

- **alias sub-element:** Historical aliases for this keyword (can appear multiple times). Alias has a single attribute name which must be lower case with underscores.

- **id:** Unique ID for the keyword, usually name with an integer appended, but not currently used/enforced.

- **minOccurs:** Minimum occurrences of the keyword in current context (set to 1 for required, 0 for optional)

- **maxOccurs:** Maximum occurrences of the keyword in current context (for example environment may appear at most once)

And optional/useful GUI-related attributes are:

- **help:** (Don’t add this attribute the new keywords!) A pointer to the corresponding reference manual section (deprecated as not needed with new reference manual format which mirrors keyword hierarchy).

- **label:** a short, friendly label string for the keyword in the GUI. Format these like titles, e.g., "Initial Point for Search".

- **group:** Category or group for this keyword, e.g., optimization vs. parameter study if they are to be groups for GUI purposes

- **Alternation** ( \(<oneOf>\): Alternation of groups of content is done with the element \(<oneOf>\) which indicates that its immediate children are alternates. In NIDR this was done with the pipe symbol: OptionA | OptionB. oneOf allows the label attribute and its use is recommended.

- **Required Group** ( \(<required>\): A required group can be specified by enclosing the contents in the \(<required>\) element. In NIDR this was done by enclosing the content in parentheses: ( required group... )

- **Optional Group** ( \(<optional>\): An optional group can be specified by enclosing the contents in the \(<optional>\) element. In NIDR this was done by enclosing the content in brackets: [ optional group... ]
3.2 Rebuild Generated Files

When configured with -DENABLE_SPEC_MAINT, performing a make in dakota.build/src will regenerate all files which derive from dakota.xml, include dakota.input.nspec, NIDR_keywds.hpp, and dakota.input- summary. If you commit changes to a source repository, be sure to commit any automatically generated files in addition to any modified in the following steps. It is not strictly necessary to run make at this point in the sequence, and in fact may generate errors if necessary handlers aren’t yet available.

Warning

Please do not manually modify generated files!

3.3 Update Parser Source NIDRProblemDescDB.cpp

Many keywords have data associated with them: an integer, a floating-point number, a string, or arrays of such entities. Data requirements are specified in dakota.input.nspec by the tokens INTEGER, REAL, STRING, INTEGRERLIST, REALLIST, STRINGLIST. (Some keywords have no associated data and hence no such token.) After each keyword and data token, the dakota.input.nspec file specifies functions that the NIDR parser should call to record the appearance of the keyword and deal with any associated data. The general form of this specification is

\{ startfcn, startdata, stopfcn, stopdata \}

i.e., a brace-enclosed list of one to four functions and data pointers, with trailing entities taken to be zero if not present; zero for a function means no function will be called. The startfcn must deal with any associated data. Otherwise, the distinction between startfcn and stopfcn is relevant only to keywords that begin a group of keywords (enclosed in parentheses or square brackets). The startfcn is called before other entities in the group are processed, and the stop function is called after they are processed. Top-level keywords often have both startfcn and stopfcn; stopfcn is uncommon but possible for lower-level keywords. The startdata and (if needed) stopdata values are usually pointers to little structures that provide keyword-specific details to generic functions for startfcn and stopfcn. Some keywords that begin groups (such as "approx_problem" within the top-level "environment" keyword) have no need of either a startfcn or a stopfcn; this is indicated by "\{0\}".

Most of the things within braces in dakota.input.nspec are invocations of macros defined in dakota.- source/src/NIDRProblemDescDB.cpp. The macros simplify writing dakota.input.nspec and make it more readable. Most macro invocations refer to little structures defined in NIDRProblemDescDB.cpp, usually with the help of other macros, some of which have different definitions in different parts of NIDRProblemDescDB.cpp. When adding a keyword to dakota.input.nspec, you may need to add a structure definition or even introduce a new data type. NIDRProblemDescDB.cpp has sections corresponding to each top-level keyword. The top-level keywords are in alphabetical order, and most entities in the section for a top-level keyword are also in alphabetical order. While not required, it is probably good practice to maintain this structure, as it makes things easier to find.

Any integer, real, or string data associated with a keyword are provided to the keyword’s startfcn, whose second argument is a pointer to a Values structure, defined in header file nidr.h.

Example 1: if you added the specification:

\[method_setting REAL \{method_setting_start, &method_setting_details\} \]

you would provide a function

```c
void NIDRProblemDescDB::
method_setting_start(const char *keyname, Values *val, void **g, void *v)
{ ... }
```

in NIDRProblemDescDB.cpp. In this example, argument &method_setting_details would be passed as v, val->n (the number of values) would be 1 and *val->r would be the REAL value given for the method- _setting keyword. The method_setting_start function would suitably store this value with the help of method_setting_details.
For some top-level keywords, $g$ (the third argument to the startfcn and stopfcn) provides access to a relevant context. For example, `method_start` (the startfcn for the top-level `method` keyword) executes

```c++
DataMethod *dm = new DataMethod;
g = (void*)dm;
```

(and supplies a couple of default values to `dm`). The start functions for lower-level keywords within the `method` keyword get access to `dm` through their $g$ arguments. Here is an example:

```c++
void NIDRProblemDescDB::
method_str(const char *keyname, Values *val, void **g, void *v)
{
    (**(DataMethod**)g)->**(String DataMethod::**)v = *val->s;
}
```

In this example, $v$ points to a pointer-to-member, and an assignment is made to one of the components of the `DataMethod` object pointed to by $g$. The corresponding stopfcn for the top-level `method` keyword is

```c++
void NIDRProblemDescDB::
method_stop(const char *keyname, Values *val, void **g, void *v)
{
    DataMethod *p = *(DataMethod**)g;
    pDDBInstance->dataMethodList.insert(*p);
    delete p;
}
```

which copies the now populated `DataMethod` object to the right place and cleans up.

**Example 2:** if you added the specification

```
[method_setting REALLIST {{N_mdm(RealL,methodCoeffs)}
```

then `method_RealL` (defined in NIDRProblemDescDB.cpp) would be called as the startfcn, and `methodCoeffs` would be the name of a (currently nonexistent) component of `DataMethod`. The `N_mdm` macro is defined in NIDRProblemDescDB.cpp; among other things, it turns `RealL` into `NIDRProblemDescDB::method_RealL`. This function is used to process lists of REAL values for several keywords. By looking at the source, you can see that the list values are `val->r[i]` for `0 <= i < val->n`.

### 3.4 Update Corresponding Data Classes

The Data classes (`DataEnvironment`, `DataMethod`, `DataModel`, `DataVariables`, `DataInterface`, and `DataResponses`) store the parsed user input data. In this step, we extend the Data class definitions to include any new attributes referred to in `dakota.xml` or `NIDRProblemDescDB`.

#### 3.4.1 Update the Data Class Header File

Add a new attribute to the public data for each of the new specifications. Follow the style guide for class attribute naming conventions (or mimic the existing code).

#### 3.4.2 Update the .cpp File

Define defaults for the new attributes in the constructor initialization list (if not a container with a sensible default constructor) in same order as they appear in the header. Add the new attributes to the `write(MPIPackBuffer&)`, `read(MPIUnpackBuffer&)`, and `write(std::ostream&)` functions, paying careful attention to the use of a consistent ordering.
3.5 Update Database Source ProblemDescDB.cpp

3.5.1 Augment/update get_<data_type>() Functions

The next update step involves extending the database retrieval functions in `dakota.source/src/ProblemDescDB.cpp`. These retrieval functions accept an identifier string and return a database attribute of a particular type, e.g., a RealVector:

```
const RealVector& get_rv(const String& entry_name);
```

The implementation of each of these functions contains tables of possible entry_name values and associated pointer-to-member values. There is one table for each relevant top-level keyword, with the top-level keyword omitted from the names in the table. Since binary search is used to look for names in these tables, each table must be kept in alphabetical order of its entry names. For example,

```
else if ((L = Begins(entry_name, "model."))) {
    if (dbRep->methodDBLocked)
        Locked_db();
#define P &DataModelRep::
    static KW<RealVector, DataModelRep> RVdmo[] = {
        // must be sorted
        {"nested.primary_response_mapping", P primaryRespCoeffs},
        {"nested.secondary_response_mapping", P secondaryRespCoeffs},
        {"surrogate.kriging_conmin_seed", P krigingConminSeed},
        {"surrogate.kriging_correlations", P krigingCorrelations},
        {"surrogate.kriging_max_correlations", P krigingMaxCorrelations},
        {"surrogate.kriging_min_correlations", P krigingMinCorrelations}{}
    #undef P
    KW<RealVector, DataModelRep> *kw;
    if ((kw = (KW<RealVector, DataModelRep>*)Binsearch(RVdmo, L)))
        return dbRep->dataModelIter->dataModelRep->*kw->p;
}
```

is the "model" portion of `ProblemDescDB::get_rv()`. Based on entry_name, it returns the relevant attribute from a `DataModel` object. Since there may be multiple model specifications, the `dataModelIter` list iterator identifies which node in the list of `DataModel` objects is used. In particular, `dataModelList` contains a list of all of the `data_model` objects, one for each time a top-level `model` keyword was seen by the parser. The particular model object used for the data retrieval is managed by `dataModelIter`, which is set in a `set_db_list_nodes()` operation that will not be described here.

There may be multiple `DataMethod`, `DataModel`, `DataVariables`, `DataInterface`, and/or `DataResponses` objects. However, only one specification is currently allowed so a list of `DataEnvironment` objects is not needed. Rather, `ProblemDescDB::environmentSpec` is the lone `DataEnvironment` object.

To augment the `get_<data_type>()` functions, add table entries with new identifier strings and pointer-to-member values that address the appropriate data attributes from the Data class object. The style for the identifier strings is a top-down hierarchical description, with specification levels separated by periods and words separated with underscores, e.g., "keyword.group_specification.individual_specification". Use the `dbRep->listIter->attribute` syntax for variables, interface, responses, and method specifications. For example, the `method.setting.example attribute` would be added to `get_rv()` as:

```
{"method_name.method.setting", P methodSetting},
```

inserted at the beginning of the `RVdmo` array shown above (since the name in the existing first entry, i.e., "nested.primary_response_mapping", comes alphabetically after "method_name.method_setting").
3.6 Use get_<data_type>() Functions

At this point, the new specifications have been mapped through all of the database classes. The only remaining step is to retrieve the new data within the constructors of the classes that need it. This is done by invoking the get_<data_type>() function on the ProblemDescDB object using the identifier string you selected in Augment/update get_<data_type>() Functions. For example:

```cpp
const String& interface_type = problem_db.get_string("interface.type");
```

passes the "interface.type" identifier string to the ProblemDescDB::get_string() retrieval function, which returns the desired attribute from the active DataInterface object.

Warning

Use of the get_<data_type>() functions is restricted to class constructors, since only in class constructors are the data list iterators (i.e., dataMethodIter, dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) guaranteed to be set correctly. Outside of the constructors, the database list nodes will correspond to the last set operation, and may not return data from the desired list node.

3.7 Update the Documentation

Doxygen comments should be added to the Data class headers for the new attributes, and the reference manual sections describing the portions of dakota.xml that have been modified should be updated by updating files in dakota.source/docs/KeywordMetaData/.ddakota.xml, together with these metadata files generates the reference manual and GUI context-aware help documentation.
Chapter 4

Understanding Iterator Flow

This page explains the various phases comprising Iterator::run_iterator(). Prior to Iterator construction, when command-line options are parsed, Boolean run mode flags corresponding to PRERUN, RUN, and POSTRUN are set in ParallelLibrary. If the user didn’t specify any specific run modes, the default is for all three to be true (all phases will execute).

Iterator is constructed. When called, run_iterator() sequences:

- initialize_run(): unconditionally called, virtual. Performs common initialization such as allocating workspaces, setting communicators and evaluation counts. When re-implementing this virtual, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.

- Not implemented: pre-run input

- IF PRERUN, invoke pre_run(): virtual function; default no-op. Purpose: derived classes should implement pre_run() if they are able to generate all parameter sets (variables) at once, separate from run(). Derived implementations should call their nearest parent’s pre_run(), typically before performing their own steps.

- IF PRERUN, invoke pre_output(): non-virtual function; if user requested, output variables to file.

- Not implemented: run input

- IF RUN, invoke virtual function run(). Purpose: at a minimum, evaluate parameter sets through computing responses; for iterators without pre/post capability, their entire implementation is in run() and this is a reasonable default for new Iterators.

- Not implemented: run output

- IF POSTRUN, invoke post_input(): virtual function, default only print helpful message on mode. Purpose: derived iterators supporting post-run input from file must implement to read file and populate variables/responses (and possibly best points) appropriately. Implementations must check if the user requested file input.

- IF POSTRUN, invoke post_run(): virtual function. Purpose: generate statistics / final results. Any analysis that can be done solely on tabular data read by post_input() can be done here. Derived re-implementations should call their nearest parent’s post-run(), typically after performing their specific post-run activities.

- Not implemented: post-run output
• `finalize_run()`: unconditionally called, virtual. Purpose: free workspaces. Default base class behavior is no-op, however, derived implementations should call their nearest parent’s `finalize_run` after performing their specialized portions.

`Iterator` is destructed.
Chapter 5

Interfacing with Dakota as a Library

5.1 Introduction

Tightly integrating or linking Dakota into another application can improve user experience by delivering a more unified, inter-operable software tool for optimization and UQ analyses, improving performance by eliminating file system-based interfaces, and reducing challenges with parallel computing inter-operation. This benefit has been realized within several Sandia and external simulation applications. This section describes how to link Dakota into another C++ application.

Dakota has two primary application programming interfaces (APIs). The LibraryEnvironment class facilitates use of Dakota as an algorithm service library within another application. In this case, the simulation application is providing a "front end" for Dakota. The second API, provided by the DirectApplicInterface class, provides an interface for Dakota to call the simulation code directly to perform function evaluations in core. This permits the simulation to be the "back end" for Dakota. The most complete library integration of Dakota would use both in combination, with the overall simulation framework providing both the front end and back end for Dakota, creating a sandwich, as loosely depicted here:

```
-----------------  
| Application     |
|                 |
| { -----        |
| { Dakota {LibraryEnvironment} |
| { { Function evaluation callback to Application (via DirectApplicInterface) |
| { { | |
| <------/ |
| { { |
| { |
| { --- |
| { --- |
| -------------
```

Attention

Dakota may be integrated as a library in other software applications subject to the terms of the GNU Lesser General Public License (LGPL). Refer to http://www.gnu.org/licenses/lgpl.html or the LICENSE file included with Dakota.

When Dakota is compiled and installed, the relevant library API headers are installed to CMAKE_INSTALL_PREFIX/include and the runtime libraries primarily to CMAKE_INSTALL_PREFIX/lib/ (on some platforms,
CHAPTER 5. INTERFACING WITH DAKOTA AS A LIBRARY

to CMAKE_INSTALL_PREFIX/bin/. The core C/C++ code is in the library dakota_src, while Fortran code lives in the dakota_src_fortran library. Information on using the API in Dakota headers is included throughout this section, while considerations for configuring and linking against Dakota and its various required and optional third–party libraries are emphasized in the section Linking against the Dakota library.

Steps involved in integrating Dakota into another application typically include:

1. Writing C++ code for your application to instantiate, configure, and execute Dakota’s LibraryEnvironment ("front end"); see Basic Dakota library instantiation and Configuring Dakota operation.

2. Writing C++ code for Dakota to call a function in your application to perform function evaluations ("back end"); see Creating a simulator plugin interface.

3. Compiling Dakota and linking into your application (Linking against the Dakota library).

Several source code examples demonstrate Dakota library interfaces. The classes SIM::SerialDirectApplicInterface and SIM::ParallelDirectApplicInterface demonstrate serial and parallel simulation function evaluation plug-ins. The file library_mode.cpp includes a main program that exercises Dakota libraries in serial and parallel modes with these mock simulator programs, with various ways of configuring Dakota problem definition and operation. Finally, library_split.cpp demonstrates running Dakota as a library modular on an MPI sub-communicator.

5.2 Basic Dakota library instantiation

The function run_dakota_parse() in library_mode.cpp demonstrates the basic use of Dakota library objects as one would in another main application that embeds Dakota. In this example, Dakota is configured based on a typical user-provided text-based Dakota input file (the same that would be provided at the command line with dakota -i dakota_optimization.in) and a function evaluator derived from a DirectApplicInterface is plugged into the Dakota library environment.

First, an object of type ProgramOptions which manages top-level Dakota settings is instantiated and configured to specify the name of the Dakota user input file. Additional options for output and error redirection, restart operation, and more may be set via ProgramOptions. See its class documentation for details.

```cpp
string dakota_input_file = "dakota_optimization.in";
Dakota::ProgramOptions opts;
opts.input_file(dakota_input_file);
```

Next, a LibraryEnvironment is created, passing the desired settings from opts:

```cpp
Dakota::LibraryEnvironment env(opts);
```

This standard constructor will parse the specified input and create Dakota objects. It assumes many default settings, including that the parent application initialized MPI if running in parallel mode. (In this case, Dakota will detect whether MPI was initialized and not call MPI_Init or MPI_Finalize.) For more advanced use cases described below, alternate constructors allow constructing based on MPI communicators, with delayed finalization, and with Dakota database update function callbacks. Then the application’s function evaluator implementing Dakota’s DirectApplicInterface is plugged in with a convenience function serial_interface_plugin() or parallel_interface_-plugin(). Finally, the Dakota analysis is run by calling

```cpp
env.execute();
```

The next two sections offer additional details on (1) alternative and supplementary ways to configure Dakota’s operation (Configuring Dakota operation) and (2) how to specialize Dakota’s DirectApplicInterface to provide a function evaluator plugin to Dakota (Creating a simulator plugin interface).
5.3. CONFIGURING DAKOTA OPERATION

Remarks

After LibraryEnvironment construction, all MPI communicator partitioning has been performed and the ParallelLibrary instance may be interrogated for parallel configuration data. For example, the lowest level communicators in Dakota’s multilevel parallel partitioning are the analysis communicators, which can be retrieved using:

```cpp
// retrieve the set of analysis communicators for simulation initialization:
// one analysis comm per ParallelConfiguration (PC), one PC per Model.
Array<MPI_Comm> analysis_comms = parallel_lib.analysis_intra_communicators();
```

These communicators can then be used for initializing parallel simulation instances when registering the plugin interface, where the number of MPI communicators in the array corresponds to one communicator per ParallelConfiguration instance. This is demonstrated below in Derivation.

5.3 Configuring Dakota operation

This section describes several alternate ways to initially set and later manipulate Dakota’s configuration, including alternatives to using a text-based input file. The algorithm configuration for a particular Dakota analysis run is managed in its ProblemDescDB, which can be populated via an input file, string literal, or C++ API, and later modified through Dakota’s C++ API. All Dakota objects then draw information from this database upon instantiation.

5.3.1 Input data parsing

The simplest way for an application to configure a Dakota analysis problem is to use Dakota’s normal input parsing system to populate its problem database (ProblemDescDB). This is done by providing standard Dakota input file syntax through the library interface, via either a file name or string literal. An advantage is that native Dakota syntax can be used, but disadvantages include the requirement for an additional input file beyond those already required by the parent application and that application users also need to know Dakota syntax.

The two ways to configure Dakota via input parsing are shown near the beginning of run_dakota_mixed() in library_mode.cpp. Here the ProgramOptions are set to either parse from a named file:

```cpp
Dakota::ProgramOptions opts;
opts.input_file(dakota_input_file);
```

or from a string literal provided by the wrapping application:

```cpp
string serial_input = "% Dakota input file ...";
opts.input_string(serial_input);
```

This library approach is coarse-grained in that input is parsed, objects constructed, and the environment is immediately ready to run. The next approaches are more modular.

5.3.2 Problem database insertion

A second approach to configuring Dakota’s operation is to bypass parsing phases and directly populate the ProblemDescDB with information on the methods, variables, interface, responses, etc., that define the Dakota analysis problem. This approach requires more interaction with Dakota classes and data structures. However, it can offer usability benefit when the integrating application does not want their users to interact with the full Dakota syntax, or efficiency benefit when for example there are a large number of variables to configure.

In the direct database population approach, Dakota DataMethod, DataModel, DataVariables, DataInterface, and DataResponses objects are instantiated and populated with the desired problem data. These objects are then
published to the problem database using \texttt{insert\_nodes}. An example of this approach is available in \texttt{run\_dakota\_data} in \texttt{library\_mode.cpp}, where the OPT++ Quasi-Newton method is configured to work on a plugin version of \texttt{text\_book} or \texttt{rosenbrock}. The data objects are populated with their default values upon instantiation and are often sufficient for basic Dakota studies. Only the non-default values need to be specified. Moreover the default Dakota Model is a SingleModel, so this object need not be configured unless tailoring its configuration or using a more advanced model type. Refer to the \texttt{DataMethod}, \texttt{DataModel}, \texttt{DataVariables}, \texttt{DataInterface}, and \texttt{DataResponses} class documentation and source code for lists of attributes and their defaults. Here is an excerpt of \texttt{run\_dakota\_data} that specifies the OPT++ solver after default construction of \texttt{DataMethod}:

\begin{verbatim}
Dakota::DataMethod dme;
Dakota::DataMethodRep* dmr = dme.data();
dmr->methodName = Dakota::OPTPP_QNEWTON;
\end{verbatim}

When using direct database population, it is critical to leave the database in an open, accessible state after initial construction. In this \texttt{run\_dakota\_data} example, a flag \texttt{check\_bcast\_construct} is passed into the \texttt{LibraryEnvironment} constructor, indicating that it should not finalize the database and construct Dakota objects. Moreover, it is only necessary to populate the database on rank 0 of the MPI Comm on which Dakota is running. After database objects are inserted or adjusted, the \texttt{LibraryEnvironment::done\_modifying\_db} function must be called before proceeding to execute. This synchronizes problem data across all ranks and constructs Dakota objects needed to run the specified analysis.

\begin{verbatim}
bool check\_bcast\_construct = false;
Dakota::LibraryEnvironment env(MPI\_COMM\_WORLD, opts, check\_bcast\_construct);
if (rank == 0) // insert/modify DB, then lock and proceed:
 env.done\_modifying\_db();
 env.execute();
\end{verbatim}

\subsection{Mixed mode, callbacks, and late updates}

The \texttt{LibraryEnvironment} API also supports mixed approaches that combine the parsing of a Dakota input file (or input string literal) with direct database updates. This approach is motivated by large-scale applications where large vectors are cumbersome to specify in a Dakota input file or where later updates to an input template are needed. The example \texttt{run\_dakota\_mixed} in \texttt{library\_mode.cpp} demonstrates the combination of these more advanced approaches: (1) input text parsing, (2) database updates via a callback, (3) database updates via direct manipulation, and (4) further runtime updates to the Model before running.

First, a \texttt{ProgramOptions} class is instantiated and configured to parse either an input file or input string literal (as in earlier examples). The passed input data must contain all required inputs so the parser can validate them. Since vector data like variable values/bounds/tags, linear/nonlinear constraint coefficients/bounds, etc., are optional, these potentially large vector specifications can be omitted from the input file and updated later through the database API. Only the variable/response counts necessary for sizing, e.g.:

\begin{verbatim}
method
 linear\_inequality\_constraints = 500

variables
 continuous\_design = 1000

responses
 objective\_functions = 1
 nonlinear\_inequality\_constraints = 100000
\end{verbatim}

and not the lists of values are required in this case. To update or add data after this initial parse, we use the \texttt{ProblemDescDB::set} family of overloaded functions, e.g.

\begin{verbatim}
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem\_db.set("variables.continuous\_design\_initial\_point", drv);
\end{verbatim}
5.3. CONFIGURING DAKOTA OPERATION

where the string identifiers are the same identifiers used when pulling information from the database using one of the get_<datatype>() functions (refer to ProblemDescDB for a full list). However, the supported ProblemDescDB::set() options are a restricted subset of the database attributes, focused on vector inputs that can be large scale.

Second, the example demonstrates a user-provided callback function which Dakota will invoke after input parsing to update ProblemDescDB. In library_mode.cpp, callback_function() is a user-provided post-parse callback that implements the type Dakota::DbCallbackFunction.

```cpp
static void callback_function(Dakota::ProblemDescDB* db, void *ptr);
```

When Dakota calls this function it will pass back pointers to the ProblemDescDB instance and to user-provided data, so the application may convey its settings by calling methods on the ProblemDescDB, optionally using the provided data. An example of a user data structure is demonstrated in callback_data. In this case, when the LibraryEnvironment is constructed, it is constructed with the input data to initially parse, the callback function, and to leave it unlocked for further updates:

```cpp
bool done_with_db = false;
Dakota::LibraryEnvironment env(opts, done_with_db,
    callback_function, &data);
```

Third, the example demonstrates changes to the database after parsing and callback-based updates. Again, these only need happen on Dakota’s rank 0 before finalizing the DB with LibraryEnvironment::done_modifying_db(). The example demonstrates:

1. Getting access to the database through env.problem_description_db()
2. Setting the database nodes to the appropriate method through problem_db.resolve_top_method()
3. Getting data from the DB with a get string array function: problem_db.get_sa("interface.application.-analysis_drivers")
4. Setting update data with problem_db.set("variables.continuous_design.initial_point", ip);

After any of these three types updates, calling LibraryEnvironment::done_modifying_db() will broadcast any updates (including potentially large vector data and post-process specification data to fill in any vector defaults that have not yet been provided through either file parsing or direct updates. (Note: scalar defaults are handled in the Data class constructors.)

Fourth and finally, run_dakota_mixed() demonstrates modifying a Model’s data after database operations and interface plugin are complete. This involves finding the right Model (or other class) instance to modify, and directly adjusting its data through the public API. Since the database is finalized, any updates must be performed through direct set operations on the constructed objects. For example, to update other data such as variable values/bounds/tags or response bounds/targets/tags, refer to the set functions documented in Iterator and Model. As an example, the following code updates the active continuous variable values, which will be employed as the initial guess for certain classes of Iterators:

```cpp
ModelList& all_models = problem_db.model_list();
Model& first_model = *all_models.begin();
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
first_model.continuous_variables(drv);
```
Remarks

If performing such data updates within the constructor of a DirectApplicInterface extension/derivation (see Creating a simulator plugin interface), then this code is sufficient since the database is unlocked, the active list nodes of the ProblemDescDB have been set for you, and the correct method/model/variables/interface/responses specification instance will get updated. The difficulty in this case stems from the order of instantiation. Since the Variables and Response instances are constructed in the base Model class, prior to construction of Interface instances in derived Model classes, database information related to Variables and Response objects will have already been extracted by the time the Interface constructor is invoked and the database update will not propagate.

Therefore, it is preferred to perform these database set operations at a higher level (e.g., within your main program), prior to allowing Environment to broadcast, construct, and execute, such that instantiation order is not an issue. However, in this case, it is necessary to explicitly manage the list nodes of the ProblemDescDB using a specification instance identifier that corresponds to an identifier from the input file, e.g.:

```c++
problem_db.set_db_variables_node("MY_VARIABLES_ID");
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem_db.set("variables.continuous_design.initial_point", drv);
```

Alternatively, rather than setting just a single data node, all data nodes may be set using a method specification identifier:

```c++
problem_db.set_db_list_nodes("MY_METHOD_ID");
```

since the method specification is responsible for identifying a model specification, which in turn identifies variables, interface, and responses specifications. If hard-wiring specification identifiers is undesirable, then

```c++
problem_db.resolve_top_method();
```

can also be used to deduce the active method specification and set all list nodes based on it. This is most appropriate in the case where only single specifications exist for method/model/variables/interface/responses. This is the approach demonstrated in run_dakota_mixed(). In each of these cases, setting list nodes unlocks the corresponding portions of the database, allowing set/get operations.

5.4 Creating a simulator plugin interface

The DirectApplicInterface class provides an interface for Dakota to call the simulation code directly to perform function evaluations mapping variables to responses. This provides the "back end" for Dakota to call back to the simulation framework. Two approaches to defining this direct interface are described here. The first is less common, while the second is recommended when possible.

5.4.1 Extension

The first approach involves extending one of the existing DirectApplicInterface subclasses (TestDriverInterface, MatlabInterface, etc.) to support additional direct simulation interfaces. For example, Dakota algebraic test problems are implemented in TestDriverInterface. One could add additional direct functions to Dakota in TestDriverInterface::derived_map_ac(). In addition, TestDriverInterface::derived_map_if() and TestDriverInterface::derived_map_of() can be extended to perform pre- and post-processing tasks if desired, but this is not required.

While this approach is the simplest, it has the disadvantage that the Dakota library will need to be recompiled when the simulation or its direct interface is modified. If it is desirable to maintain the independence of the Dakota library from the host application, then the derivation approach described in the next section should be employed.
5.4. CREATING A SIMULATOR PLUGIN INTERFACE

Remarks

If the new direct evaluation function implementation will not be a member function of one of the Dakota classes, then the following prototype should be used in order to pass the required data:

```c++
int sim(const Dakota::Variables& vars, const Dakota::ActiveSet& set, 
        Dakota::Response& response);
```

If the new function will be a member function, e.g., in `TestDriverInterface`, then this can be simplified to

```c++
int sim();
```

since the data access can be performed through the `DirectApplicInterface` class attributes.}

5.4.2 Derivation

The second approach is to derive a new interface from `DirectApplicInterface` and redefine several virtual functions. As demonstrated in `SIM::SerialDirectApplicInterface` and `SIM::ParallelDirectApplicInterface`, a typical derived class declaration might be

```c++
namespace SIM {

class SerialDirectApplicInterface: public Dakota::DirectApplicInterface
{
public:

    // Constructor and destructor
    SerialDirectApplicInterface(const Dakota::ProblemDescDB& problem_db);

protected:

    // Virtual function redefinitions
    int derived_map_if(const Dakota::String& if_name);
    int derived_map_ac(const Dakota::String& ac_name);
    int derived_map_of(const Dakota::String& of_name);

private:

    // Data
}
} // namespace SIM
```

where the new derived class resides in the simulation’s namespace. Similar to the case of `Extension`, the `DirectApplicInterface::derived_map_ac()` function is the required redefinition, and `DirectApplicInterface::derived_map_if()` and `DirectApplicInterface::derived_map_of()` are optional.

Typically the new `derived_map_ac()` implementation delegates to the main simulation application for a function evaluation. Here Dakota variables would get mapped into the simulation’s data structures, the simulation executed, and derived response data computed for return to Dakota.

Once a derived application class is created, it must be plugged in, or registered, with the appropriate `Interface` in the `LibraryEnvironment`. In MPI cases where Dakota is potentially managing concurrent evaluations of the simulation, the plugin must be configured to run on the right MPI sub-communicator, or Dakota analysis comm. The simpler case is demonstrated in `serial_interface_plugin.cpp` in `library_mode.cpp`, while a more advanced case using the analysis communicator is shown in `parallel_interface_plugin.cpp`.

The Dakota `LibraryEnvironment` provides a convenience function to plugin an `Interface`. This example will replace any interface found matching the given model, interface, and analysis driver with the passed plugin interface:
std::string model_type(""); // demo: empty string will match any model type
dstd::string interf_type("direct");
std::string an_driver("plugin_rosenbrook");
Dakota::ProblemDescDB problem_db = env.problem_description_db();
Dakota::Interface* serial_iface =
    new SIM::SerialDirectApplicInterface(problem_db);
bool plugged_in =
    env.plugin_interface(model_type, interf_type, an_driver, serialiface);

The LibraryEnvironment also provides convenience functions that allow the client to iterate the lists of available interfaces or models for more advanced cases. For instance if the client knows there is only a single interface active, it could get the list of available interfaces of length 1 and plugin to the first one. In the more advanced case where the simulation interface instance should manage parallel simulations within the context of an MPI communicator, one should pass in the relevant analysis communicator(s) to the derived constructor. For the latter case of looping over a set of models, the simplest approach of passing a single analysis communicator would use code similar to

Dakota::Modellist filt_models =
    env.filtered_modellist("single", "direct", "plugin_text_book");
Dakota::ProblemDescDB problem_db = env.problem_description_db();
Dakota::Modelliter ml_iter;
for (ml_iter = filt_models.begin(); ml_iter != filt_models.end(); ++ml_iter) {
    // set DB nodes to input specification for this Model
    problem_db.set_db_model_nodes(ml_iter->model_id());
    Dakota::Interface& model_interface = ml_iter->derived_interface();
    // Parallel case: plug in derived Interface object with an analysisComm.
    // Note: retrieval and passing of analysisComm is necessary only if
    // parallel operations will be performed in the derived constructor.
    // retrieve the currently active analysisComm from the Model. In the most
    // general case, need an array of Comms to cover all Model configurations.
    const MPI_Comm& analysis_comm = ml_iter->analysis_comm();
    // don’t increment ref count since no other envelope shares this letter
    model_interface.assign_rep(new SIM::ParallelDirectApplicInterface(problem_db, analysis_comm), false);
}

The file library_mode.cpp demonstrates each of these approaches. Since a Model may be used in multiple parallel contexts and may therefore have a set of parallel configurations, a more general approach would extract and pass an array of analysis communicators to allow initialization for each of the parallel configurations.

New derived direct interface instances inherit various attributes of use in configuring the simulation. In particular, the ApplicationInterface::parallelLib reference provides access to MPI communicator data (e.g., the analysis communicators discussed above), DirectApplicInterface::analysisDrivers provides the analysis driver names specified by the user in the input file, and DirectApplicInterface::analysisComponents provides additional analysis component identifiers (such as mesh file names) provided by the user which can be used to distinguish different instances of the same simulation interface. It is worth noting that inherited attributes that are set as part of the parallel configuration (instead of being extracted from the ProblemDescDB) will be set to their defaults following construction of the base class instance for the derived class plug-in. It is not until run-time (i.e., within derived_map_if/derived_map_ac/derived_map_of) that the parallel configuration settings are re-propagated to the plug-in instance. This is the reason that the analysis communicator should be passed in to the constructor of a parallel plug-in, if the constructor will be responsible for parallel application initialization.
5.5 Retrieving data after a run

After executing the Dakota Environment, final results can be obtained through the use of Environment::variables_results() and Environment::response_results(), e.g.:

```cpp
// retrieve the final parameter values
const Variables& vars = env.variables_results();

// retrieve the final response values
const Response& resp = env.response_results();
```

In the case of optimization, the final design is returned, and in the case of uncertainty quantification, the final statistics are returned. Dakota has a prototype results database, which will eventually provide better access to the results from a study.

5.6 Linking against the Dakota library

This section presumes Dakota has been configured with CMake, compiled, and installed to a CMAKE_INSTALL_PREFIX using make install or equivalent. The Dakota libraries against which you must link will typically install to CMAKE_INSTALL_PREFIX/bin/ and CMAKE_INSTALL_PREFIX/lib/, while headers are provided in CMAKE_INSTALL_PREFIX/include/. The core Dakota C and C++ code is in the library dakota-src, while Fortran code lives in the dakota-src_fortran library. Runtime libraries for any configure-enabled Dakota third-party software components (such as DOT, NPSOL, OPT++, LHS, etc.) are also installed to the lib/ directory. Applications link against these Dakota libraries by specifying appropriate include and link directives.

There are two primary ways to determine the necessary Dakota-related libraries and link order for linking your application. First, when running CMake, a list of required Dakota and Dakota-included third-party libraries will be output to the console, e.g.,

```
-- Dakota_LIBRARIES: dakota_src;dakota_src_fortran;nidr;teuchos;pecos;pecos_src;lhs;mods;mod;fftpack;sparselapack
```

While external dependencies will be output as:

```
-- Dakota_TPL_LIBRARIES: /usr/lib64/libcurl.so;/usr/lib64/openmpi/lib/libm_pi_cxx.so;debug;/usr/lib64/libbz.so;
```

Note that depending on how you configured Dakota, some libraries may be omitted from these lists (for example commercial add-ons NPSOL, DOT, and NLPLQL), or additional libraries may appear.

A second option is to check which libraries appear in CMAKE_INSTALL_PREFIX/bin/, CMAKE_INSTALL_PREFIX/lib/, or more accurately, see the file Makefile.export.Dakota in the Dakota build/src/ or installation include/ directory. Here are some additional notes on specific libraries:

- Some Boost libraries (boost_regex, boost_filesystem, boost_system, boost_serialization) are required, and other Boost library components may be required depending on configuration, e.g., boost_signals when configuring with HAVE_ACRO BOOL=TRUE

- System compiler and math libraries may need to be included, as may additional system libraries such as Expat and Curl, depending on how Dakota is configured.

- If configuring with graphics, you will need to add the dakota_scilplot library and system X libraries (partial list here):

  ```
  -lxpm -lxm -lxv -lxmu -lxp -lxext -lxI1 -lsm -lXCE
  ```
CHAPTER 5. INTERFACING WITH DAKOTA AS A LIBRARY

- When configuring with AMPL (HAVEAMPL:BOOL=ON), the AMPL solver library may require dl, funcadd0.o and fl libraries. We have experienced problems with the creation of libamplsolver.a on some platforms; use the dakota-users mailing list to get help with any problems related to this.

- Optional library GSL (discouraged due to GPL license) and if linking with system-provided GSL, gslcblas may be needed if Dakota was configured with them.

- Newmat: as of Dakota 5.2, -lnewmat is no longer required

Finally, it is important to use the same C++ compiler (possibly an MPI wrapper) for compiling Dakota and your application and potentially include Dakota-related preprocessor defines as emitted by CMake during compilation of Dakota and included in Makefile.export.Dakota. This ensures that the platform configuration settings are properly synchronized across Dakota and your application.
Chapter 6

Performing Function Evaluations

Performing function evaluations is one of the most critical functions of the Dakota software. It can also be one of the most complicated, as a variety of scheduling approaches and parallelism levels are supported. This complexity manifests itself in the code through a series of cascaded member functions, from the top level model evaluation functions, through various scheduling routines, to the low level details of performing a system call, fork, or direct function invocation. This section provides an overview of the primary classes and member functions involved.

6.1 Synchronous function evaluations

For a synchronous (i.e., blocking) mapping of parameters to responses, an iterator invokes Model::evaluate() to perform a function evaluation. This function is all that is seen from the iterator level, as underlying complexities are isolated. The binding of this top level function with lower level functions is as follows:

- Model::evaluate() utilizes Model::derived_evaluate() for portions of the response computation specific to derived model classes.
- Model::derived_evaluate() directly or indirectly invokes Interface::map().
- Interface::map() utilizes ApplicationInterface::derived_map() for portions of the mapping specific to derived application interface classes.

6.2 Asynchronous function evaluations

For an asynchronous (i.e., nonblocking) mapping of parameters to responses, an iterator invokes Model::evaluate_nowait() multiple times to queue asynchronous jobs and then invokes either Model::synchronize() or Model::synchronize_nowait() to schedule the queued jobs in blocking or nonblocking fashion. Again, these functions are all that is seen from the iterator level, as underlying complexities are isolated. The binding of these top level functions with lower level functions is as follows:

- Model::evaluate_nowait() utilizes Model::derived_evaluate_nowait() for portions of the response computation specific to derived model classes.
- This derived model class function directly or indirectly invokes Interface::map() in asynchronous mode, which adds the job to a scheduling queue.
- Model::synchronize() or Model::synchronize_nowait() utilize Model::derived_synchronize() or Model::derived_synchronize_nowait() for portions of the scheduling process specific to derived model classes.
CHAPTER 6. PERFORMING FUNCTION EVALUATIONS

- These derived model class functions directly or indirectly invoke Interface::synch() or Interface::synch_nowait().

- For application interfaces, these interface synchronization functions are responsible for performing evaluation scheduling in one of the following modes: master dynamic, peer dynamic or peer static.

- NOTE: The Interface evaluation scheduling in Dakota was refactored for releases 5.4 and 6.0. Discussion of the new Interface-related functions is currently missing here.

6.3 Analyses within each function evaluation

NOTE: The Interface evaluation scheduling in Dakota was refactored for releases 5.4 and 6.0. Discussion of the new Interface-related functions for analyses is currently missing here.

The discussion above covers the parallelism level of concurrent function evaluations serving an iterator. For the parallelism level of concurrent analyses serving a function evaluation, similar schedulers are involved to support synchronous local, asynchronous local, message passing, and hybrid modes. Not all of the schedulers are elevated to the ApplicationInterface level since the system call and direct function interfaces do not yet support nonblocking local analyses (and therefore support synchronous local and message passing modes, but not asynchronous local or hybrid modes). Fork interfaces, however, support all modes of analysis parallelism.
Chapter 7

Working with Variable Containers and Views

Variable views control the subset of variable types that are active and inactive within a particular iterative study. For design optimization and uncertainty quantification (UQ), for example, the active variables view consists of design or uncertain types, respectively, and any other variable types are carried along invisible to the iterative algorithm being employed. For parameter studies and design of experiments, however, a variable subset view is not imposed and all variables are active. Selected UQ methods can also be toggled into an "All" view using the `active all` variables input specification. When not in an All view, finer gradations within the uncertain variable sets are also relevant: probabilistic methods (reliability, stochastic expansion) view aleatory uncertain variables as active, nonprobabilistic methods (interval, evidence) view epistemic uncertain variables as active, and a few UQ methods (sampling) view both as active. In a more advanced NestedModel use case such as optimization under uncertainty, design variables are active in the outer optimization context and the uncertain variables are active in the inner UQ context, with an additional requirement on the inner UQ level to return derivatives with respect to its "inactive" variables (i.e., the design variables) for use in the outer optimization loop.

For efficiency, contiguous arrays of data store variable information for each of the domain types (continuous, discrete integer, and discrete real), but active and inactive views into them permit selecting subsets in a given context. This management is encapsulated into the Variables and SharedVariablesData classes. This page clarifies concepts of relaxed (formerly merged) vs. mixed, fine-grained vs. aggregated types, domain types, and views into contiguous arrays.

We begin with an overview of the storage and management concept, for which the following two sections describe the storage of variable values and meta-data about their organization, used in part to manage views. They are intended to communicate rationale to maintainers of Variables and SharedVariablesData classes. The final section provides a discussion of active and inactive views.

### 7.1 Storage in Variables

As described in the Main Page Variables, a Variables object manages variable types (design, aleatory uncertain, epistemic uncertain, and state) and domain types (continuous, discrete integer, and discrete real) and supports different approaches to either distinguishing among these types or aggregating them. Two techniques are used in cooperation to accomplish this management: (1) class specialization (RelaxedVariables or MixedVariables) and (2) views into contiguous variable arrays. The latter technique is used whenever it can satisfy the requirement, with fallback to class specialization when it cannot. In particular, aggregation or separation of variable types can be accomplished with views, but for aggregation or separation of variable domains, we must resort to class
specialization in order to relax discrete domain types. In this class specialization, a RelaxedVariables object combines continuous and discrete types (relaxing integers to reals) whereas a MixedVariables object maintains the integer/real distinction throughout.

The core data for a Variables instance is stored in a set of three contiguous arrays, corresponding to the domain types: allContinuousVars, allDiscreteIntVars, and allDiscreteRealVars, unique to each Variables instance.

Within the core variable data arrays, data corresponding to different aggregated variable types are stored in sequence for each domain type:

- continuous: [design, aleatory uncertain, epistemic uncertain, state]
- discrete integer: [design, aleatory uncertain, (epistemic uncertain), state]
- discrete real: [design, aleatory uncertain, (epistemic uncertain), state]

Note there are currently no epistemic discrete variables. This domain type ordering (continuous, discrete integer, discrete real) and aggregated variable type ordering (design, aleatory uncertain, epistemic uncertain, state) is preserved whenever distinct types are flattened into single contiguous arrays. Note that the aleatory and epistemic uncertain variables contain sub-types for different distributions (e.g., normal, uniform, histogram, poisson), and discrete integer types include both integer ranges and integer set sub-types. All sub-types are ordered according to their order of appearance in dakota.input.nspec.

When relaxing in MixedVariables, the allContinuousVars will also aggregate the discrete types, such that they contain ALL design, then ALL uncertain, then ALL state variables, each in aggregated type order; the allDiscrete-IntVars and allDiscreteRealVars arrays are empty.

7.2 Storage in SharedVariablesData

Each Variables instance contains a reference-counted SharedVariablesData object that stores information on the variables configuration. This configuration data includes counts, types, IDs, and labels, which are often the same across many Variables instances. Thus, SharedVariablesData is intended to reduce the memory footprint by allowing the sharing of a single copy of redundant information among different Variables instances.

One of the purposes of this shared information is to support mappings between variable types, IDs, and indices into the storage arrays. Variable "types" refer to the fine-grained variable types a user would specify in an input file, as enumerated in DataVariables.hpp, e.g., CONTINUOUS, DESIGN, WEIBULL, UNCERTAIN, DISCRETE, STATE, RANGE, etc. variablesComponents is a map from these variable types to counts of how many are present.

In contrast, the variablesCompsTotals array stores total counts of each "aggregated type" (design, aleatory uncertain, epistemic uncertain, state) which might be selected to be active in a given view. Thus this array has length 12 to track the combinations of three domain type storage arrays with four possible aggregated variable types: \{continuous, discrete integer, discrete real\} x \{design, aleatory uncertain, epistemic uncertain, state\}. For example, the first entry of this array stores the number of continuous design variables, the second the number of discrete integer design (including both discrete design range and discrete design set integer types), and the last the number of discrete real state variables.

The arrays allContinuousTypes, allDiscreteIntTypes, and allDiscreteRealTypes are sized to match the corresponding core domain type storage arrays. They track the fine-grained variable type stored in that entry of the data array (since when relaxed, the continuous array may be storing data corresponding to discrete data).

Finally allContinuousIds stores the 1-based IDs of the variables stored in the allContinuousVars array, i.e., the variable number of all the problem variables considered as a single contiguous set, in aggregate type order. For relaxed (formerly merged) views, relaxedDiscreteIds stores the 1-based IDs of the variables which have been relaxed into the continuous array.
7.3. ACTIVE AND INACTIVE VIEWS

These counts, types, and IDs are most commonly used within the Model classes for mappings between variables objects at different levels of a model recursion. See, for example, the variable mappings in the NestedModel constructor.

7.3 Active and inactive views

The pair SharedVariablesDataRep::variablesView tracks the active and inactive views of the data, with values taken from the enum in DataVariables.hpp. The valid values include EMPTY and the combinations \{relaxed, mixed\} x \{all, design, aleatory uncertain, epistemic uncertain, uncertain, state\}. The ALL cases indicate aggregation of the design, aleatory uncertain, epistemic uncertain, and state types, whereas the DISTINCT cases indicate either no aggregation (design, aleatory uncertain, epistemic uncertain, state) or reduced aggregation (aleatory+epistemic uncertain). The active view is determined by the algorithm in use, managed in Variables::get_view(). Any inactive view is set based on higher level iteration within a model recursion (e.g., a NestedModel), which enables lower level iteration to return derivatives with respect to variables that are active at the higher level. In the case where there is no higher level iteration, then the inactive view will remain EMPTY. It is important to stress that "inactive" at one level corresponds to active at another, and therefore the inactive set of variables should not be interpreted as the strict complement of the active set of variables; rather, active and inactive are both subsets whose union may still be a subset of the total container (more precise terminology might involve "primary" active and "secondary" active or similar). An active complement view could potentially be supported in the future, should the need arise, although this view would require management of non-contiguous portions of the aggregated arrays.

Given these groupings (views), the active and inactive subsets of the allContinuousVars, allDiscreteIntVars, and allDiscreteRealVars arrays are always contiguous, permitting vector views of the underlying data using either Teuchos::View (for numerical vectors) or Boost.MultiArray (for book-keeping arrays) views.

When a Variables envelope is constructed, its letter is initialized to either a RelaxedVariables or Mixed-Variables object depending on the active view. The derived classes size the contiguous storage arrays to accommodate all the problem variables, and then initialize active views into them, which could involve either subsets (DISTINCT active views) or views of the full arrays (ALL active views). Inactive views, on the other hand, are initialized during construction of a model recursion (e.g., a call to Model::inactive_view() in the NestedModel constructor). Thus, active variable subsets are always available but inactive variable subsets will be EMPTY prior to them being initialized within a Model recursion.

Accessors for continuous variables include:

- continuous_variables(): returns the active view which might return all (ALL views) or a subset (DISTINCT views) such as design, uncertain, only aleatory uncertain, etc.
- inactive_continuous_variables(): returns the inactive view which is either a subset or empty
- all_continuous_variables(): returns the full vector allContinuousVars

and this pattern is followed for active/inactive/all access to discrete_int_variables() and discrete_real_variables() as well as for labels, IDs, and types in SharedVariablesData and variable bounds in Constraints.

Member SurfpackApproximation::build (size_t index=_NPOS)

Right now, we’re completely deleting the old data and then recopying the current data into a SurfData object. This was just the easiest way to arrive at a solution that would build and run. This function is frequently called from addPoint rebuild, however, and it’s not good to go through this whole process every time one more data point is added.

Member SurfpackApproximation::hessian (const RealVector &c_vars)

Make this acceptably efficient
Chapter 7. Working with Variable Containers and Views

Member SurfpackApproximation::hessian (const Variables &vars)

  Make this acceptably efficient
Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

- **Dakota**
  - The primary namespace for DAKOTA ........................................... 75

- **SIM**
  - A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA 163
Chapter 9

Hierarchical Index

9.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ActiveSet ................................................. 165
Approximation ........................................... 202
  GaussProcApproximation ............................... 367
  PecosApproximation .................................. 700
  SurfpackApproximation ............................... 834
  TANA3Approximation ................................. 861
  TaylorApproximation ................................. 863
  VPSApproximation ................................... 890
APPSEvalMgr ............................................. 216
AproproWriter .......................................... 219
BaseConstructor ......................................... 220
BootstrapSamplerBase< Data > ......................... 222
  BootstrapSampler< Data > ............................ 220
  BootstrapSamplerWithGS< Data, Getter, Setter > .... 224
BootstrapSamplerBase< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > > .... 222
  BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > > .... 221
callback_data ........................................... 225
COLINApplication ....................................... 225
CommandShell ........................................... 232
ConsoleRedirector ...................................... 237
Constraints .............................................. 239
  MixedVarConstraints ................................. 440
  RelaxedVarConstraints .............................. 760
DataEnvironment ....................................... 248
DataEnvironmentRep ................................... 249
DataInterface ........................................... 266
DataMethod .............................................. 267
DataMethodRep .......................................... 268
DataModel ................................................ 284
DataModelRep ............................................ 285
<table>
<thead>
<tr>
<th>Class/Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataResponses</td>
<td>291</td>
</tr>
<tr>
<td>DataResponsesRep</td>
<td>293</td>
</tr>
<tr>
<td>DataVariables</td>
<td>302</td>
</tr>
<tr>
<td>DataVariablesRep</td>
<td>304</td>
</tr>
<tr>
<td>DiscrepancyCorrection</td>
<td>325</td>
</tr>
<tr>
<td>JEGAOptimizer::Driver</td>
<td>331</td>
</tr>
<tr>
<td>Environment</td>
<td>338</td>
</tr>
<tr>
<td>ExecutableEnvironment</td>
<td>350</td>
</tr>
<tr>
<td>LibraryEnvironment</td>
<td>426</td>
</tr>
<tr>
<td>NomadOptimizer::Evaluator</td>
<td>342</td>
</tr>
<tr>
<td>JEGAOptimizer::Evaluator</td>
<td>344</td>
</tr>
<tr>
<td>JEGAOptimizer::EvaluatorCreator</td>
<td>349</td>
</tr>
<tr>
<td>ExperimentData</td>
<td>350</td>
</tr>
<tr>
<td>FileReadException</td>
<td>360</td>
</tr>
<tr>
<td>ResultsFileError</td>
<td>775</td>
</tr>
<tr>
<td>TabularDataTruncated</td>
<td>859</td>
</tr>
<tr>
<td>FunctionEvalFailure</td>
<td>366</td>
</tr>
<tr>
<td>GeneralReader</td>
<td>371</td>
</tr>
<tr>
<td>GeneralWriter</td>
<td>372</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>372</td>
</tr>
<tr>
<td>CommandLineHandler</td>
<td>231</td>
</tr>
<tr>
<td>Graphics</td>
<td>375</td>
</tr>
<tr>
<td>Interface</td>
<td>386</td>
</tr>
<tr>
<td>ApplicationInterface</td>
<td>186</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
<td>321</td>
</tr>
<tr>
<td>MatlabInterface</td>
<td>429</td>
</tr>
<tr>
<td>PythonInterface</td>
<td>740</td>
</tr>
<tr>
<td>ScilabInterface</td>
<td>789</td>
</tr>
<tr>
<td>TestDriverInterface</td>
<td>864</td>
</tr>
<tr>
<td>ParallelDirectApplicInterface</td>
<td>673</td>
</tr>
<tr>
<td>SerialDirectApplicInterface</td>
<td>797</td>
</tr>
<tr>
<td>ProcessApplicInterface</td>
<td>722</td>
</tr>
<tr>
<td>ProcessHandleApplicInterface</td>
<td>726</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>360</td>
</tr>
<tr>
<td>SpawnApplicInterface</td>
<td>833</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
<td>856</td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>377</td>
</tr>
<tr>
<td>ApproximationInterface</td>
<td>209</td>
</tr>
<tr>
<td>Iterator</td>
<td>395</td>
</tr>
<tr>
<td>Analyzer</td>
<td>179</td>
</tr>
<tr>
<td>NonD</td>
<td>506</td>
</tr>
<tr>
<td>NonDCalibration</td>
<td>536</td>
</tr>
<tr>
<td>NonDBayesCalibration</td>
<td>527</td>
</tr>
<tr>
<td>NonDDREAMBayesCalibration</td>
<td>540</td>
</tr>
<tr>
<td>NonDQUESTOBayesCalibration</td>
<td>621</td>
</tr>
<tr>
<td>NonDGPMSABayesCalibration</td>
<td>562</td>
</tr>
<tr>
<td>NonDWASABIBayesCalibration</td>
<td>648</td>
</tr>
</tbody>
</table>
### 9.1. CLASS HIERARCHY

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonDExpansion</td>
<td>544</td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>611</td>
</tr>
<tr>
<td>NonDMultilevelPolynomialChaos</td>
<td>594</td>
</tr>
<tr>
<td>NonDStochCollocation</td>
<td>645</td>
</tr>
<tr>
<td>NonDMultilevelStochCollocation</td>
<td>606</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>566</td>
</tr>
<tr>
<td>NonDCubature</td>
<td>537</td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>617</td>
</tr>
<tr>
<td>NonDSparseGrid</td>
<td>642</td>
</tr>
<tr>
<td>NonDInterval</td>
<td>569</td>
</tr>
<tr>
<td>NonDGlobalInterval</td>
<td>552</td>
</tr>
<tr>
<td>NonDGlobalEvidence</td>
<td>550</td>
</tr>
<tr>
<td>NonDGlobalSingleInterval</td>
<td>558</td>
</tr>
<tr>
<td>NonDLHSInterval</td>
<td>573</td>
</tr>
<tr>
<td>NonDLHSEvidence</td>
<td>572</td>
</tr>
<tr>
<td>NonDLHSSingleInterval</td>
<td>578</td>
</tr>
<tr>
<td>NonDLocalInterval</td>
<td>580</td>
</tr>
<tr>
<td>NonDLocalEvidence</td>
<td>579</td>
</tr>
<tr>
<td>NonDLocalSingleInterval</td>
<td>593</td>
</tr>
<tr>
<td>NonDPOFDarts</td>
<td>608</td>
</tr>
<tr>
<td>NonDReliability</td>
<td>627</td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
<td>555</td>
</tr>
<tr>
<td>NonDLocalReliability</td>
<td>583</td>
</tr>
<tr>
<td>NonDRKDDarts</td>
<td>629</td>
</tr>
<tr>
<td>NonDSampling</td>
<td>632</td>
</tr>
<tr>
<td>NonDAdaptImpSampling</td>
<td>517</td>
</tr>
<tr>
<td>NonDAdaptiveSampling</td>
<td>521</td>
</tr>
<tr>
<td>NonDGPImpSampling</td>
<td>559</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>574</td>
</tr>
<tr>
<td>NonDMultilevelSampling</td>
<td>597</td>
</tr>
<tr>
<td>PStudyDACE</td>
<td>735</td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>317</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>362</td>
</tr>
<tr>
<td>ParamStudy</td>
<td>690</td>
</tr>
<tr>
<td>PSUADesignCompExp</td>
<td>737</td>
</tr>
<tr>
<td>Verification</td>
<td>887</td>
</tr>
<tr>
<td>RichExtrapVerification</td>
<td>779</td>
</tr>
<tr>
<td>MetaIterator</td>
<td>431</td>
</tr>
<tr>
<td>CollabHybridMetaIterator</td>
<td>229</td>
</tr>
<tr>
<td>ConcurrentMetaIterator</td>
<td>233</td>
</tr>
<tr>
<td>EmbedHybridMetaIterator</td>
<td>336</td>
</tr>
<tr>
<td>SeqHybridMetaIterator</td>
<td>793</td>
</tr>
<tr>
<td>Minimizer</td>
<td>432</td>
</tr>
<tr>
<td>LeastSq</td>
<td>423</td>
</tr>
<tr>
<td>Optimizer</td>
<td>662</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>415</td>
</tr>
<tr>
<td>NonlinearCGOptimizer</td>
<td>651</td>
</tr>
<tr>
<td>NOWPACOptimizer</td>
<td>656</td>
</tr>
</tbody>
</table>
# CHAPTER 9. HIERARCHICAL INDEX

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptDartsOptimizer</td>
<td>659</td>
</tr>
<tr>
<td>SurrBasedMinimizer</td>
<td>847</td>
</tr>
<tr>
<td>EffGlobalMinimizer</td>
<td>333</td>
</tr>
<tr>
<td>SurrBasedGlobalMinimizer</td>
<td>838</td>
</tr>
<tr>
<td>SurrBasedLocalMinimizer</td>
<td>840</td>
</tr>
<tr>
<td>IteratorScheduler</td>
<td>409</td>
</tr>
<tr>
<td>LabelsWriter</td>
<td>422</td>
</tr>
<tr>
<td>LightWtBaseConstructor</td>
<td>428</td>
</tr>
<tr>
<td>MatchesWC</td>
<td>429</td>
</tr>
<tr>
<td>Model</td>
<td>443</td>
</tr>
<tr>
<td>NestedModel</td>
<td>484</td>
</tr>
<tr>
<td>RecastModel</td>
<td>749</td>
</tr>
<tr>
<td>ActiveSubspaceModel</td>
<td>167</td>
</tr>
<tr>
<td>AdaptedBasisModel</td>
<td>174</td>
</tr>
<tr>
<td>DataTransformModel</td>
<td>297</td>
</tr>
<tr>
<td>ProbabilityTransformModel</td>
<td>704</td>
</tr>
<tr>
<td>RandomFieldModel</td>
<td>744</td>
</tr>
<tr>
<td>ScalingModel</td>
<td>782</td>
</tr>
<tr>
<td>SimulationModel</td>
<td>823</td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>851</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
<td>252</td>
</tr>
<tr>
<td>HierarchSurrModel</td>
<td>380</td>
</tr>
<tr>
<td>MPIManager</td>
<td>478</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>479</td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>481</td>
</tr>
<tr>
<td>NL2Res</td>
<td>500</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>504</td>
</tr>
<tr>
<td>NOWPACBlackBoxEvaluator</td>
<td>655</td>
</tr>
<tr>
<td>OutputManager</td>
<td>667</td>
</tr>
<tr>
<td>OutputWriter</td>
<td>670</td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>671</td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>675</td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>677</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>687</td>
</tr>
<tr>
<td>partial_prr_equalilty</td>
<td>696</td>
</tr>
<tr>
<td>partial_prr_hash</td>
<td>696</td>
</tr>
<tr>
<td>PebblBranching</td>
<td>696</td>
</tr>
<tr>
<td>PebblBranchSub</td>
<td>697</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>711</td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
<td>495</td>
</tr>
<tr>
<td>ProgramOptions</td>
<td>730</td>
</tr>
<tr>
<td>QuesoJointPdf&lt; V, M&gt;</td>
<td>742</td>
</tr>
<tr>
<td>QuesoVectorRV&lt; V, M&gt;</td>
<td>743</td>
</tr>
<tr>
<td>ReducedBasis</td>
<td>759</td>
</tr>
<tr>
<td>Response</td>
<td>763</td>
</tr>
<tr>
<td>ExperimentResponse</td>
<td>358</td>
</tr>
<tr>
<td>SimulationResponse</td>
<td>826</td>
</tr>
<tr>
<td>RestartWriter</td>
<td>770</td>
</tr>
</tbody>
</table>
9.1. CLASS HIERARCHY

ResultsDBAny ................................................................. 771
ResultsEntry < StoredType > ............................................ 774
ResultsID ................................................................. 775
ResultsManager .......................................................... 776
ResultsNames ............................................................ 778
ScalingOptions ............................................................ 788
SensAnalysisGlobal ....................................................... 790
TrackerHTTP::Server ..................................................... 798
SharedApproxData ......................................................... 798
    SharedPecosApproxData ........................................... 804
    SharedSurfpackApproxData ...................................... 811
SharedResponseData ..................................................... 806
SharedResponseDataRep .................................................. 809
SharedVariablesData .................................................... 812
SharedVariablesDataRep ................................................ 818
SNLLBase ................................................................. 827
SOLBase ................................................................. 831
TabularReader ............................................................ 860
TabularWriter ............................................................ 860
TrackerHTTP .............................................................. 871
TraitsBase ............................................................... 873
    AppsTraits .......................................................... 218
    COLINTraits ........................................................ 228
    CONMINTraits ...................................................... 236
    DataFitSurrBasedLocalTraits .................................... 251
    DL SolverTraits ..................................................... 329
    DOTTraits .......................................................... 330
    EffGlobalTraits ..................................................... 335
    HierarchSurrBasedLocalTraits .................................. 379
    JEGA Traits ........................................................ 421
    NCSU Traits ........................................................ 484
    NL2SOLLeastSqTraits .............................................. 501
    NLPQLP Traits ........................................................ 502
    NLSSOLLeastSqTraits .............................................. 503
    Nomad Traits ........................................................ 504
    NonlinearCG Traits ................................................. 654
    NOWPAC Traits ...................................................... 657
    NPSOL Traits ........................................................ 658
    OptDarts Traits ...................................................... 662
    Pebblid Traits ...................................................... 699
    SNLLLeastSqTraits ................................................ 829
    SNLL Traits ........................................................ 830
    SurrBasedGlobal Traits .......................................... 839
UsageTracker ............................................................. 875
Var_ijcheck .............................................................. 876
Var_rcheck .............................................................. 877
Variables ............................................................... 877
    MixedVariables ...................................................... 441
    RelaxedVariables ................................................... 761
CHAPTER 9. HIERARCHICAL INDEX

VLint ......................................................... 888
VLreal ......................................................... 889
VLstr ......................................................... 889
WorkdirHelper .............................................. 894
Chapter 10

Class Index

10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

**ActiveSet**
Container class for active set tracking information. Contains the active set request vector and the derivative variables vector .................................................... 165

**ActiveSubspaceModel**
Active subspace model for input (variable space) reduction ........................................ 167

**AdaptedBasisModel**
Adapted basis model for input (variable space) reduction ........................................ 174

**Analyzer**
Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy .......... 179

**ApplicationInterface**
Derived class within the interface class hierarchy for supporting interfaces to simulation codes 186

**Approximation**
Base class for the approximation class hierarchy .................................................... 202

**ApproximationInterface**
Derived class within the interface class hierarchy for supporting approximations to simulation-based results ................................................................. 209

**APPSEvalMgr**
Evaluation manager class for APPSPACK ............................................................. 216

**AppsTraits**
Wrapper class for HOPSPACK ................................................................................... 218

**ApreproWriter**
Utility used in derived write_core to write in aprepro format ................................... 219

**BaseConstructor**
Dummy struct for overloading letter-envelope constructors ....................................... 220

**BootstrapSampler< Data >**
Actual bootstrap sampler implementation for common data types ......................... 220

**BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >**
Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix .......... 221

**BootstrapSamplerBase< Data >**
Base class/interface for the bootstrap sampler ......................................................... 222
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BootstrapSamplerWithGS&lt;Data, Getter, Setter&gt;</td>
<td>A derived sampler to allow for user specification of the accessor methods</td>
<td>224</td>
</tr>
<tr>
<td>callback_data</td>
<td></td>
<td>225</td>
</tr>
<tr>
<td>COLINApplication</td>
<td></td>
<td>225</td>
</tr>
<tr>
<td>COLINTraits</td>
<td>Wrapper class for optimizers defined using COLIN</td>
<td>228</td>
</tr>
<tr>
<td>CollabHybridMetaIterator</td>
<td>Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods</td>
<td>229</td>
</tr>
<tr>
<td>CommandLineHandler</td>
<td>Utility class for managing command line inputs to DAKOTA</td>
<td>231</td>
</tr>
<tr>
<td>CommandShell</td>
<td>Utility class which defines convenience operators for spawning processes with system calls</td>
<td>232</td>
</tr>
<tr>
<td>ConcurrentMetaIterator</td>
<td>Meta-iterator for multi-start iteration or pareto set optimization</td>
<td>233</td>
</tr>
<tr>
<td>CONMINTraits</td>
<td>Wrapper class for the CONMIN optimization library</td>
<td>236</td>
</tr>
<tr>
<td>ConsoleRedirector</td>
<td></td>
<td>237</td>
</tr>
<tr>
<td>Constraints</td>
<td>Base class for the variable constraints class hierarchy</td>
<td>239</td>
</tr>
<tr>
<td>DataEnvironment</td>
<td>Handle class for environment specification data</td>
<td>248</td>
</tr>
<tr>
<td>DataEnvironmentRep</td>
<td>Body class for environment specification data</td>
<td>249</td>
</tr>
<tr>
<td>DataFitSurrBasedLocalTraits</td>
<td>Class for provably-convergent local surrogate-based optimization and nonlinear least squares</td>
<td>251</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
<td>Derived model class within the surrogate model branch for managing data fit surrogates (global and local)</td>
<td>252</td>
</tr>
<tr>
<td>DataInterface</td>
<td>Handle class for interface specification data</td>
<td>266</td>
</tr>
<tr>
<td>DataMethod</td>
<td>Handle class for method specification data</td>
<td>267</td>
</tr>
<tr>
<td>DataMethodRep</td>
<td>Body class for method specification data</td>
<td>268</td>
</tr>
<tr>
<td>DataModel</td>
<td>Handle class for model specification data</td>
<td>284</td>
</tr>
<tr>
<td>DataModelRep</td>
<td>Body class for model specification data</td>
<td>285</td>
</tr>
<tr>
<td>DataResponses</td>
<td>Handle class for responses specification data</td>
<td>291</td>
</tr>
<tr>
<td>DataResponsesRep</td>
<td>Body class for responses specification data</td>
<td>293</td>
</tr>
<tr>
<td>DataTransformModel</td>
<td>Data transformation specialization of RecastModel</td>
<td>297</td>
</tr>
<tr>
<td>DataVariables</td>
<td>Handle class for variables specification data</td>
<td>302</td>
</tr>
<tr>
<td>DataVariablesRep</td>
<td>Body class for variables specification data</td>
<td>304</td>
</tr>
</tbody>
</table>
10.1. CLASS LIST

DDACEDesignCompExp
Wrapper class for the DDACE design of experiments library ........................................... 317

DirectApplicInterface
Derived application interface class which spawns simulation codes and testers using direct
procedure calls .................................................................................................................. 321

DiscrepancyCorrection
Base class for discrepancy corrections .............................................................................. 325

DLSolverTraits
A version of TraitsBase specialized for DLSolver .......................................................... 329

DOTTraits
Wrapper class for the DOT optimization library .............................................................. 330

JEGAOptimizer::Driver
A subclass of the JEGA front end driver that exposes the individual protected methods to
execute the algorithm .................................................................................................... 331

EffGlobalMinimizer ................................................................. 333

EffGlobalTraits
Implementation of Efficient Global Optimization/Least Squares algorithms .................. 335

EmbedHybridMetaIterator
Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local
search methods within global search methods .................................................................. 336

Environment
Base class for the environment class hierarchy ............................................................ 338

NomadOptimizer::Evaluator
NOMAD-based Evaluator class ..................................................................................... 342

JEGAOptimizer::Evaluator
An evaluator specialization that knows how to interact with Dakota ............................... 344

JEGAOptimizer::EvaluatorCreator
A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a
Evaluator ....................................................................................................................... 349

ExecutableEnvironment
Environment corresponding to execution as a stand-alone application ........................ 350

ExperimentData
Interpolation method for interpolating between experimental and model data. I need to work
on inputs/outputs to this method. For now, this assumes interpolation of functional data ... 350

ExperimentResponse
Container class for response functions and their derivatives. ExperimentResponse provides
the body class .................................................................................................................. 358

FileReadException
Base class for Dakota file read exceptions (to allow catching both tabular and general file
truncation issues) .......................................................................................................... 360

ForkApplicInterface
Derived application interface class which spawns simulation codes using fork/execvp/waitpid 360

FSUDesignCompExp
Wrapper class for the FSUDace QMC/CVT library ......................................................... 362

FunctionEvalFailure
Exception class for function evaluation failures .................................................................. 366

GaussProcApproximation
Derived approximation class for Gaussian Process implementation ............................. 367
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GeneralReader</strong></td>
<td>Utility used in derived read_core to read in generic format</td>
<td>371</td>
</tr>
<tr>
<td><strong>GeneralWriter</strong></td>
<td>Utility used in derived write_core to write in generic format</td>
<td>372</td>
</tr>
<tr>
<td><strong>GetLongOpt</strong></td>
<td>GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France)</td>
<td>372</td>
</tr>
<tr>
<td><strong>Graphics</strong></td>
<td>Single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this</td>
<td>375</td>
</tr>
<tr>
<td><strong>OutputManager::dakotaGraphics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GridApplicInterface</strong></td>
<td>Derived application interface class which spawns simulation codes using grid services such as Condor or Globus</td>
<td>377</td>
</tr>
<tr>
<td><strong>HierarchSurrBasedLocalTraits</strong></td>
<td>Class for multilevel-multifidelity optimization algorithm</td>
<td>379</td>
</tr>
<tr>
<td><strong>HierarchSurrModel</strong></td>
<td>Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity)</td>
<td>380</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>Base class for the interface class hierarchy</td>
<td>386</td>
</tr>
<tr>
<td><strong>Iterator</strong></td>
<td>Base class for the iterator class hierarchy</td>
<td>395</td>
</tr>
<tr>
<td><strong>IteratorScheduler</strong></td>
<td>This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g., meta-iteration, nested models)</td>
<td>409</td>
</tr>
<tr>
<td><strong>JEGAOptimizer</strong></td>
<td>A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA)</td>
<td>415</td>
</tr>
<tr>
<td><strong>JEGATraits</strong></td>
<td>A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA)</td>
<td>421</td>
</tr>
<tr>
<td><strong>LabelsWriter</strong></td>
<td>Utility used in derived write_core to write labels in tabular format</td>
<td>422</td>
</tr>
<tr>
<td><strong>LeastSq</strong></td>
<td>Base class for the nonlinear least squares branch of the iterator hierarchy</td>
<td>423</td>
</tr>
<tr>
<td><strong>LibraryEnvironment</strong></td>
<td>Environment corresponding to execution as an embedded library</td>
<td>426</td>
</tr>
<tr>
<td><strong>LightWtBaseConstructor</strong></td>
<td>Dummy struct for overloading constructors used in on-the-fly Model instantiations</td>
<td>428</td>
</tr>
<tr>
<td><strong>MatchesWC</strong></td>
<td>Predicate that returns true when the passed path matches the wild card with which it was configured. Currently supports * and ?</td>
<td>429</td>
</tr>
<tr>
<td><strong>MatlabInterface</strong></td>
<td></td>
<td>429</td>
</tr>
<tr>
<td><strong>MetaIterator</strong></td>
<td>Base class for meta-iterators</td>
<td>431</td>
</tr>
<tr>
<td><strong>Minimizer</strong></td>
<td>Base class for the optimizer and least squares branches of the iterator hierarchy</td>
<td>432</td>
</tr>
<tr>
<td><strong>MixedVarConstraints</strong></td>
<td>Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging)</td>
<td>440</td>
</tr>
</tbody>
</table>
10.1. CLASS LIST

MixedVariables
    Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging) ................................................. 441

Model
    Base class for the model class hierarchy .................................................. 443

MPIManager
    Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD ................................................................. 478

MPIPackBuffer
    Class for packing MPI message buffers ..................................................... 479

MPIUnpackBuffer
    Class for unpacking MPI message buffers ................................................ 481

NCSUTraits
    Wrapper class for the NCSU DIRECT optimization library ............................ 484

NestedModel
    Derived model class which performs a complete sub-iterator execution within every evaluation of the model ................................................................. 484

NIDRProblemDescDB
    The derived input file database utilizing the new IDR parser ....................... 495

NL2Res
    Auxiliary information passed to calcr and calcj via ur .............................. 500

NL2SOLLeastSqTraits
    Wrapper class for the NL2SOL nonlinear least squares library ...................... 501

NLPQLPTraits
    Wrapper class for the NLPQLP optimization library, Version 2.0 .................. 502

NLSSOLLeastSqTraits
    Wrapper class for the NLSSOL nonlinear least squares library ...................... 503

NoDBBaseConstructor
    Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support ................................................................. 504

NomadTraits
    Wrapper class for NOMAD Optimizer ......................................................... 504

NonD
    Base class for all nondeterministic iterators (the DAKOTA/UQ branch) ........... 506

NonDAadaptImpSampling
    Class for the Adaptive Importance Sampling methods within DAKOTA ............. 517

NonDAadaptiveSampling
    Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate ........................................ 521

NonDBayesCalibration
    Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data ............................................................... 527

NonDCalibration ................................................................. 536

NonDCubature
    Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals .......................... 537

NonDdreamBayesCalibration
    Bayesian inference using the DREAM approach ........................................ 540
CHAPTER 10. CLASS INDEX

NonDExpansion
  Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC) . . . . . 544

NonDGGlobalEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ . . . . . . . 550

NonDGGlobalInterval
  Class for using global non-gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification . . . . . . . . . . . . . . . . . . . . 552

NonDGGlobalReliability
  Class for global reliability methods within DAKOTA/UQ . . . . . . . . . . . . . . . . . . . 555

NonDGGlobalSingleInterval
  Class for using global non-gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification . . . . . . . . . . . . . . . . . . . . 558

NonDGImpSampling
  Class for the Gaussian Process-based Importance Sampling method . . . . . . . . . . . . . 559

NonDGPMASABayesCalibration
  Generates posterior distribution on model parameters given experiment data . . . . . . . . 562

NonDIntegration
  Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals . . . . . . . . . . . . . . . . . . . . . 566

NonDInterval
  Base class for interval-based methods within DAKOTA/UQ . . . . . . . . . . . . . . . . . 569

NonDLHSEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ . . . . . . . 572

NonDLHSInterval
  Class for the LHS-based interval methods within DAKOTA/UQ . . . . . . . . . . . . . . . 573

NonDLHSSampling
  Performs LHS and Monte Carlo sampling for uncertainty quantification . . . . . . . . . . 574

NonDLHSSingleInterval
  Class for pure interval propagation using LHS . . . . . . . . . . . . . . . . . . . . . . . . 578

NonDLocalEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ . . . . . . . 579

NonDLocalInterval
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification . . . . . . . . . . . . . . . . . . . . 580

NonDLocalReliability
  Class for the reliability methods within DAKOTA/UQ . . . . . . . . . . . . . . . . . . . . 583

NonDLocalSingleInterval
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification . . . . . . . . . . . . . . . . . . . . 593

NonDMultilevelPolynomialChaos
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification . . . . 594

NonDMultilevelSampling
  Performs Multilevel Monte Carlo sampling for uncertainty quantification . . . . . . . . 597

NonDMultilevelStochCollocation
  Nonintrusive stochastic collocation approaches to uncertainty quantification . . . . . . . 606

NonDOFMDarts
  Base class for POF Dart methods within DAKOTA/UQ . . . . . . . . . . . . . . . . . . . 608

NonDPolynomialChaos
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification . . . . 611
10.1. CLASS LIST

NonDQuadrature
Derived nondeterministic class that generates N-dimensional numerical quadrature points for
evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas
617

NonDQUESTorchBayesCalibration
Bayesian inference using the QUESO library from UT Austin .......................... 621

NonDReliability
Base class for the reliability methods within DAKOTA/UQ .............................. 627

NonDRKDDArts
Base class for the Recursive k-d Dart methods within DAKOTA/UQ .................. 629

NonDSampling
Base class for common code between NonDLHSSampling, NonDAdaptImpSampling, and
other specializations ......................................................... 632

NonDSparseGrid
Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numer-
ical evaluation of expectation integrals over independent standard random variables ........ 642

NonDStochCollocation
Nonintrusive stochastic collocation approaches to uncertainty quantification ............ 645

NonDWAStorBayesCalibration
WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference ....... 648

NonlinearCGOptimizer ...................................................... 651

NonlinearCGTraits
A version of TraitsBase specialized for NonlinearCG optimizers ...................... 654

NOWPACBlackBoxEvaluator
Derived class for plugging Dakota evaluations into NOWPAC solver .................. 655

NOWPACOptimizer
Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT) 656

NOWPACTraits
A version of TraitsBase specialized for NOWPAC optimizers .......................... 657

NPSOLTraits
Wrapper class for the NPSOL optimization library ....................................... 658

OptDartsOptimizer
Wrapper class for OptDarts Optimizer ............................................. 659

OptDartsTraits
A version of TraitsBase specialized for OptDarts ....................................... 662

Optimizer
Base class for the optimizer branch of the iterator hierarchy ......................... 662

OutputManager
Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank
0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and
delegate to Graphics for X Windows Graphics ..................................... 667

OutputWriter ................................................................. 670

ParallelConfiguration
Container class for a set of ParallelLevel list iterators that collectively identify a particular
multilevel parallel configuration ........................................... 671

ParallelDirectApplicInterface
Sample derived interface class for testing parallel simulator plug-ins using assign_rep() .... 673

ParallelLevel
Container class for the data associated with a single level of communicator partitioning .... 675
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParallelLibrary</td>
<td>Class for partitioning multiple levels of parallelism and managing message passing within these levels</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>Container class for a variables object, a response object, and an evaluation id</td>
</tr>
<tr>
<td>ParamStudy</td>
<td>Class for vector, list, centered, and multidimensional parameter studies</td>
</tr>
<tr>
<td>partial_prp_equality</td>
<td>Predicate for comparing ONLY the interfaceId and Vars attributes of PRPair</td>
</tr>
<tr>
<td>partial_prp_hash</td>
<td>Wrapper to delegate to the ParamResponsePair hash_value function</td>
</tr>
<tr>
<td>PebbldBranching</td>
<td>Main Branching class for the PEBBL-based Minimizer</td>
</tr>
<tr>
<td>PebbldBranchSub</td>
<td>Sub Branch class for the PEBBL-based Minimizer</td>
</tr>
<tr>
<td>PebbldTraits</td>
<td>Wrapper class for experimental PebbldMinimizer</td>
</tr>
<tr>
<td>PecosApproximation</td>
<td>Derived approximation class for global basis polynomials</td>
</tr>
<tr>
<td>ProbabilityTransformModel</td>
<td>Probability transformation specialization of RecastModel</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>The database containing information parsed from the DAKOTA input file</td>
</tr>
<tr>
<td>ProcessApplicInterface</td>
<td>Derived application interface class that spawns a simulation code using a separate process and communicates with it through files</td>
</tr>
<tr>
<td>ProcessHandleApplicInterface</td>
<td>Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files</td>
</tr>
<tr>
<td>ProgramOptions</td>
<td>ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag()</td>
</tr>
<tr>
<td>PStudyDACE</td>
<td>Base class for managing common aspects of parameter studies and design of experiments methods</td>
</tr>
<tr>
<td>PSUADEDesignCompExp</td>
<td>Wrapper class for the PSUADE library</td>
</tr>
<tr>
<td>PythonInterface</td>
<td></td>
</tr>
<tr>
<td>QuesoJointPdf&lt; V, M &gt;</td>
<td>Dakota specialization of QUESO generic joint PDF</td>
</tr>
<tr>
<td>QuesoVectorRV&lt; V, M &gt;</td>
<td>Dakota specialization of QUESO vector-valued random variable</td>
</tr>
<tr>
<td>RandomFieldModel</td>
<td>Random field model, capable of generating and then forward propagating</td>
</tr>
<tr>
<td>RecastModel</td>
<td>Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs</td>
</tr>
<tr>
<td>ReducedBasis</td>
<td></td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RelaxedVarConstraints</td>
<td>Derived class within the <code>Constraints</code> hierarchy which employs relaxation of discrete variables</td>
</tr>
<tr>
<td>RelaxedVariables</td>
<td>Derived class within the <code>Variables</code> hierarchy which employs the relaxation of discrete variables</td>
</tr>
<tr>
<td>Response</td>
<td>Container class for response functions and their derivatives. <code>Response</code> provides the enveloper base class</td>
</tr>
<tr>
<td>RestartWriter</td>
<td></td>
</tr>
<tr>
<td>ResultsDBAny</td>
<td></td>
</tr>
<tr>
<td>ResultsEntry&lt; StoredType &gt;</td>
<td>Class to manage in-core vs. file database lookups</td>
</tr>
<tr>
<td>ResultsFileError</td>
<td>Exception throw for other results file read error</td>
</tr>
<tr>
<td>ResultsID</td>
<td>Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each <code>Iterator::run()</code> call creates or increments this count for its string identifier</td>
</tr>
<tr>
<td>ResultsManager</td>
<td>Results manager for iterator final data</td>
</tr>
<tr>
<td>ResultsNames</td>
<td>List of valid names for iterator results</td>
</tr>
<tr>
<td>RichExtrapVerification</td>
<td>Class for Richardson extrapolation for code and solution verification</td>
</tr>
<tr>
<td>ScalingModel</td>
<td>Scaling specialization of <code>RecastModel</code></td>
</tr>
<tr>
<td>ScalingOptions</td>
<td>Simple container for user-provided scaling data, possibly expanded by replicates through the models</td>
</tr>
<tr>
<td>ScilabInterface</td>
<td></td>
</tr>
<tr>
<td>SensAnalysisGlobal</td>
<td>Class for a utility class containing correlation calculations and variance-based decomposition</td>
</tr>
<tr>
<td>SeqHybridMetaIterator</td>
<td>Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity</td>
</tr>
<tr>
<td>SerialDirectApplicInterface</td>
<td>Sample derived interface class for testing serial simulator plug-ins using <code>assign_rep()</code></td>
</tr>
<tr>
<td>TrackerHTTP::Server</td>
<td>Struct to hold tracker/proxy pairs</td>
</tr>
<tr>
<td>SharedApproxData</td>
<td>Base class for the shared approximation data class hierarchy</td>
</tr>
<tr>
<td>SharedPecosApproxData</td>
<td>Derived approximation class for global basis polynomials</td>
</tr>
<tr>
<td>SharedResponseData</td>
<td>Container class encapsulating variables data that can be shared among a set of <code>Response</code> instances</td>
</tr>
<tr>
<td>SharedResponseDataRep</td>
<td>The representation of a <code>SharedResponseData</code> instance. This representation, or body, may be shared by multiple <code>SharedResponseData</code> handle instances</td>
</tr>
</tbody>
</table>
### CHAPTER 10. CLASS INDEX

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SharedSurfpackApproxData</td>
<td>Derived approximation class for Surfpack approximation classes. <strong>Interface</strong> between Surfpack and Dakota</td>
</tr>
<tr>
<td>SharedVariablesData</td>
<td>Container class encapsulating variables data that can be shared among a set of Variables instances</td>
</tr>
<tr>
<td>SharedVariablesDataRep</td>
<td>The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances</td>
</tr>
<tr>
<td>SimulationModel</td>
<td>Derived model class which utilizes a simulation-based application interface to map variables into responses</td>
</tr>
<tr>
<td>SimulationResponse</td>
<td>Container class for response functions and their derivatives. SimulationResponse provides the body class</td>
</tr>
<tr>
<td>SNLLBase</td>
<td>Base class for OPT++ optimization and least squares methods</td>
</tr>
<tr>
<td>SNLLLeastSqTraits</td>
<td>Wrapper class for the OPT++ optimization library</td>
</tr>
<tr>
<td>SNLLTraits</td>
<td>Wrapper class for the OPT++ optimization library</td>
</tr>
<tr>
<td>SOLBase</td>
<td>Base class for Stanford SOL software</td>
</tr>
<tr>
<td>SpawnApplicInterface</td>
<td>Derived application interface class which spawns simulation codes using spawnvp</td>
</tr>
<tr>
<td>SurfpackApproximation</td>
<td>Derived approximation class for Surfpack approximation classes. <strong>Interface</strong> between Surfpack and Dakota</td>
</tr>
<tr>
<td>SurrBasedGlobalMinimizer</td>
<td>Class for provably-convergent local surrogate-based optimization and nonlinear least squares</td>
</tr>
<tr>
<td>SurrBasedLocalMinimizer</td>
<td>Base class for local/global surrogate-based optimization/least squares</td>
</tr>
<tr>
<td>SurrBasedMinimizer</td>
<td>Base class for surrogate models (DataFitSurrModel and HierarchSurrModel)</td>
</tr>
<tr>
<td>SurrBasedGlobalTraits</td>
<td>The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls</td>
</tr>
<tr>
<td>TabularDataTruncated</td>
<td>Exception thrown when data read truncated</td>
</tr>
<tr>
<td>TabularReader</td>
<td>Utility used in derived read_core to read values in tabular format</td>
</tr>
<tr>
<td>TabularWriter</td>
<td>Utility used in derived write_core to write values in tabular format</td>
</tr>
<tr>
<td>TANA3Approximation</td>
<td>Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation)</td>
</tr>
</tbody>
</table>
### 10.1. CLASS LIST

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaylorApproximation</td>
<td>Derived approximation class for first- or second-order Taylor series (a local approximation)</td>
<td>863</td>
</tr>
<tr>
<td>TestDriverInterface</td>
<td></td>
<td>864</td>
</tr>
<tr>
<td>TrackerHTTP</td>
<td>TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library</td>
<td>871</td>
</tr>
<tr>
<td>TraitsBase</td>
<td>Base class for traits</td>
<td>873</td>
</tr>
<tr>
<td>UsageTracker</td>
<td>Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL</td>
<td>875</td>
</tr>
<tr>
<td>Var_i_check</td>
<td>Structure for verifying bounds and initial point for string-valued vars</td>
<td>876</td>
</tr>
<tr>
<td>Var_r_check</td>
<td>Structure for verifying bounds and initial point for real-valued vars</td>
<td>877</td>
</tr>
<tr>
<td>Variables</td>
<td>Base class for the variables class hierarchy</td>
<td>877</td>
</tr>
<tr>
<td>Verification</td>
<td>Base class for managing common aspects of verification studies</td>
<td>887</td>
</tr>
<tr>
<td>VLint</td>
<td>Structure for validating integer uncertain variable labels, bounds, values</td>
<td>888</td>
</tr>
<tr>
<td>VLreal</td>
<td>Structure for validating real uncertain variable labels, bounds, values</td>
<td>889</td>
</tr>
<tr>
<td>VLstr</td>
<td>Structure for validating string uncertain variable labels, bounds, values</td>
<td>889</td>
</tr>
<tr>
<td>VPSApproximation</td>
<td>Derived approximation class for VPS implementation</td>
<td>890</td>
</tr>
<tr>
<td>WorkdirHelper</td>
<td></td>
<td>894</td>
</tr>
</tbody>
</table>
Chapter 11

File Index

11.1 File List

Here is a list of all documented files with brief descriptions:

- **dakota_dll_api.cpp**
  This file contains a DakotaRunner class, which launches DAKOTA .................. 901

- **dakota_dll_api.h**
  API for DLL interactions ......................................................... 902

- **dakota_linear_algebra.hpp**
  Dakota linear algebra utilities ................................................ 903

- **dakota_tabular_io.hpp**
  Utility functions for reading and writing tabular data files Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data_util.h ........................................... 904

- **dll_tester.cpp**
  Test the DLL with a DAKOTA input file ......................................... 906

- **JEGAOptimizer.cpp**
  Contains the implementation of the JEGAOptimizer class ....................... 906

- **JEGAOptimizer.hpp**
  Contains the definition of the JEGAOptimizer class .............................. 907

- **library_mode.cpp**
  File containing a mock simulator main for testing Dakota in library mode .......... 908

- **library_split.cpp**
  File containing a mock simulator main for testing DAKOTA in library mode on a split communicator ................................. 911

- **main.cpp**
  File containing the main program for DAKOTA .................................... 912

- **restart_util.cpp**
  File containing the DAKOTA restart utility main program .......................... 912
Chapter 12

Namespace Documentation

12.1 Dakota Namespace Reference

The primary namespace for DAKOTA.

Classes

- class ActiveSubspaceModel
  Active subspace model for input (variable space) reduction.
- class AdaptedBasisModel
  Adapted basis model for input (variable space) reduction.
- class ApplicationInterface
  Derived class within the interface class hierarchy for supporting interfaces to simulation codes.
- class ApproximationInterface
  Derived class within the interface class hierarchy for supporting approximations to simulation-based results.
- class APPSEvalMgr
  Evaluation manager class for APPSPACK.
- class AppsTraits
  Wrapper class for HOPSPACK.
- class BootstrapSamplerBase
  Base class/interface for the bootstrap sampler.
- class BootstrapSampler
  Actual bootstrap sampler implementation for common data types.
- class BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >
  Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix.
- class BootstrapSamplerWithGS
  A derived sampler to allow for user specification of the accessor methods.
- class COLINApplication
- class COLINTraits
  Wrapper class for optimizers defined using COLIN.
- class CollabHybridMetaIterator
  Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.
• class GetLongOpt
  
  GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

• class CommandLineHandler
  
  Utility class for managing command line inputs to DAKOTA.

• class CommandShell
  
  Utility class which defines convenience operators for spawning processes with system calls.

• class ConcurrentMetaIterator
  
  Meta-iterator for multi-start iteration or pareto set optimization.

• class CONMINTraits
  
  Wrapper class for the CONMIN optimization library.

• class FileReadException
  
  base class for Dakota file read exceptions (to allow catching both tabular and general file truncation issues)

• class TabularDataTruncated
  
  exception thrown when data read truncated

• class ResultsFileError
  
  exception throw for other results file read error

• class FunctionEvalFailure
  
  exception class for function evaluation failures

• struct BaseConstructor
  
  Dummy struct for overloading letter-envelope constructors.

• struct NoDBBaseConstructor
  
  Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support.

• struct LightWtBaseConstructor
  
  Dummy struct for overloading constructors used in on-the-fly Model instantiations.

• class ActiveSet
  
  Container class for active set tracking information. Contains the active set request vector and the derivative variables vector.

• class Analyzer
  
  Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy.

• class Approximation
  
  Base class for the approximation class hierarchy.

• class Constraints
  
  Base class for the variable constraints class hierarchy.

• class Environment
  
  Base class for the environment class hierarchy.

• class Graphics
  
  The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this OutputManager::dakotaGraphics.

• class Interface
  
  Base class for the interface class hierarchy.

• class Iterator
  
  Base class for the iterator class hierarchy.

• class LeastSq
12.1. DAKOTA NAMESPACE REFERENCE

Base class for the nonlinear least squares branch of the iterator hierarchy.

- class Minimizer
  Base class for the optimizer and least squares branches of the iterator hierarchy.

- class ScalingOptions
  Simple container for user-provided scaling data, possibly expanded by replicates through the models.

- class Model
  Base class for the model class hierarchy.

- class NonD
  Base class for all nondeterministic iterators (the DAKOTA/UQ branch).

- class Optimizer
  Base class for the optimizer branch of the iterator hierarchy.

- class PStudyDACE
  Base class for managing common aspects of parameter studies and design of experiments methods.

- class Response
  Container class for response functions and their derivatives. Response provides the enveloper base class.

- class TraitsBase
  Base class for traits.

- class GeneralReader
  Utility used in derived read_core to read in generic format.

- class TabularReader
  Utility used in derived read_core to read values in tabular format.

- class GeneralWriter
  Utility used in derived write_core to write in generic format.

- class ApreproWriter
  Utility used in derived write_core to write in aprepro format.

- class TabularWriter
  Utility used in derived write_core to write values in tabular format.

- class LabelsWriter
  Utility used in derived write_core to write labels in tabular format.

- class Variables
  Base class for the variables class hierarchy.

- class Verification
  Base class for managing common aspects of verification studies.

- class DataEnvironmentRep
  Body class for environment specification data.

- class DataEnvironment
  Handle class for environment specification data.

- class DataFitSurfBasedLocalTraits
  Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

- class DataFitSurfModel
  Derived model class within the surrogate model branch for managing data fit surrogates (global and local).

- class DataInterface
  Handle class for interface specification data.
• class DataMethodRep
  Body class for method specification data.
• class DataMethod
  Handle class for method specification data.
• class DataModelRep
  Body class for model specification data.
• class DataModel
  Handle class for model specification data.
• class DataResponsesRep
  Body class for responses specification data.
• class DataResponses
  Handle class for responses specification data.
• class DataTransformModel
  Data transformation specialization of RecastModel.
• class DataVariablesRep
  Body class for variables specification data.
• class DataVariables
  Handle class for variables specification data.
• class DDACEDesignCompExp
  Wrapper class for the DDACE design of experiments library.
• class DirectApplicInterface
  Derived application interface class which spawns simulation codes and testers using direct procedure calls.
• class DiscrepancyCorrection
  Base class for discrepancy corrections.
• class DLSolverTraits
  A version of TraitsBase specialized for DLSolver.
• class DOTTraits
  Wrapper class for the DOT optimization library.
• class EffGlobalTraits
  Implementation of Efficient Global Optimization/Least Squares algorithms.
• class EffGlobalMinimizer
• class EmbedHybridMetaIterator
  Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.
• class ExecutableEnvironment
  Environment corresponding to execution as a stand-alone application.
• class ExperimentData
  Interpolation method for interpolating between experimental and model data. I need to work on inputs/outputs to this method. For now, this assumes interpolation of functional data.
• class ExperimentResponse
  Container class for response functions and their derivatives. ExperimentResponse provides the body class.
• class ForkApplicInterface
  Derived application interface class which spawns simulation codes using fork/execvp/waitpid.
12.1. DAKOTA NAMESPACE REFERENCE

- `class FSUDesignCompExp`
  Wrapper class for the FSUDace QMC/CVT library.
- `class GaussProcApproximation`
  Derived approximation class for Gaussian Process implementation.
- `class GridApplicInterface`
  Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.
- `class HierarchSurrBasedLocalTraits`
  Class for multilevel-multifidelity optimization algorithm.
- `class HierarchSurrModel`
  Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).
- `class IteratorScheduler`
  This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g., meta-iteration, nested models).
- `class JEGAOptimizer`
  A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).
- `class JEGATraits`
  A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).
- `class LibraryEnvironment`
  Environment corresponding to execution as an embedded library.
- `class MatlabInterface`
- `class MetaIterator`
  Base class for meta-iterators.
- `class MixedVarConstraints`
  Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging).
- `class MixedVariables`
  Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).
- `class MPIManager`
  Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.
- `class MPIPackBuffer`
  Class for packing MPI message buffers.
- `class MPIUnpackBuffer`
  Class for unpacking MPI message buffers.
- `class NCSUTraits`
  Wrapper class for the NCSU DIRECT optimization library.
- `class NestedModel`
  Derived model class which performs a complete sub-iterator execution within every evaluation of the model.
- `struct Var_rcheck`
  Structure for verifying bounds and initial point for real-valued vars
- `struct Var_icheck`
  Structure for verifying bounds and initial point for string-valued vars
- `struct VLreal`
structure for validating real uncertain variable labels, bounds, values

• struct VLInt
  structure for validating integer uncertain variable labels, bounds, values

• struct VLstr
  structure for validating string uncertain variable labels, bounds, values

• class NIDRProblemDescDB
  The derived input file database utilizing the new IDR parser.

• struct NL2Res
  Auxiliary information passed to calcr and calcj via ur.

• class NL2SOLLastSqTraits
  Wrapper class for the NL2SOL nonlinear least squares library.

• class NLQPPLPTrails
  Wrapper class for the NLQPPLP optimization library, Version 2.0.

• class NLSSOLLastSqTraits
  Wrapper class for the NLSSOL nonlinear least squares library.

• class NomadTraits
  Wrapper class for NOMAD Optimizer.

• class NonDAdaptImpSampling
  Class for the Adaptive Importance Sampling methods within DAKOTA.

• class NonDAdaptiveSampling
  Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

• class NonDBayesCalibration
  Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.

• class NonDCalibration

• class NonDCubature
  Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

• class NonDDREAMBayesCalibration
  Bayesian inference using the DREAM approach.

• class NonDExpansion
  Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC)

• class NonDGlobalEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

• class NonDGlobalInterval
  Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

• class NonDGlobalReliability
  Class for global reliability methods within DAKOTA/UQ.

• class NonDGlobalSingleInterval
  Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

• class NonDGImpSampling
  Class for the Gaussian Process-based Importance Sampling method.
• class NonDGPMsABayesCalibration
  Generates posterior distribution on model parameters given experiment data.
• class NonDIntegration
  Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.
• class NonDInterval
  Base class for interval-based methods within DAKOTA/UQ.
• class NonDLHSEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.
• class NonDInterval
  Base class for interval-based methods within DAKOTA/UQ.
• class NonDLHSSampling
  Performs LHS and Monte Carlo sampling for uncertainty quantification.
• class NonDInterval
  Class for pure interval propagation using LHS.
• class NonDLocalEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.
• class NonDLocalInterval
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.
• class NonDLocalReliability
  Class for the reliability methods within DAKOTA/UQ.
• class NonDLocalInterval
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.
• class NonDMultilevelPolynomialChaos
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.
• class NonDMultilevelSampling
  Performs Multilevel Monte Carlo sampling for uncertainty quantification.
• class NonDMultilevelStochCollocation
  Nonintrusive stochastic collocation approaches to uncertainty quantification.
• class NonDPOFDarts
  Base class for POF Dart methods within DAKOTA/UQ.
• class NonDPolynomialChaos
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.
• class NonDQuadrature
  Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.
• class QuesoJointPdf
  Dakota specialization of QUESO generic joint PDF.
• class QuesoVectorRV
  Dakota specialization of QUESO vector-valued random variable.
• class NonDQuesoBayesCalibration
  Bayesian inference using the QUESO library from UT Austin.
• class NonDReliability
  Base class for the reliability methods within DAKOTA/UQ.
• class NonDRKDarts
  Base class for the Recursive k-d Dart methods within DAKOTA/UQ.
• class NonDSampling
  Base class for common code between NonDLHSSampling, NonDAdaptImpSampling, and other specializations.
• class NonDSparseGrid
  Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.
• class NonDStochCollocation
  Nonintrusive stochastic collocation approaches to uncertainty quantification.
• class NonDWasABIBayesCalibration
  WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference.
• class NonlinearCGTraits
  A version of TraitsBase specialized for NonlinearCG optimizers.
• class NonlinearCGOptimizer
• class NOWPACBlackBoxEvaluator
  Derived class for plugging Dakota evaluations into NOWPAC solver.
• class NOWPACTraits
  A version of TraitsBase specialized for NOWPAC optimizers.
• class NOWPACOptimizer
  Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT)
• class NPSOLTraits
  Wrapper class for the NPSOL optimization library.
• class OptDartsTraits
  A version of TraitsBase specialized for OptDarts.
• class OptDartsOptimizer
  Wrapper class for OptDarts Optimizer.
• class OutputWriter
• class ConsoleRedirector
• class RestartWriter
• class OutputManager
  Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank 0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and delegate to Graphics for X Windows Graphics.
• class ParallelLevel
  Container class for the data associated with a single level of communicator partitioning.
• class ParallelConfiguration
  Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration.
• class ParallelILibrary
  Class for partitioning multiple levels of parallelism and managing message passing within these levels.
• class ParamResponsePair
  Container class for a variables object, a response object, and an evaluation id.
12.1. DAKOTA NAMESPACE REFERENCE

- **class ParamStudy**
  
  Class for vector, list, centered, and multidimensional parameter studies.

- **class PebbldBranching**
  
  Main Branching class for the PEBBL-based Minimizer.

- **class PebbldBranchSub**
  
  Sub Branch class for the PEBBL-based Minimizer.

- **class PebbldTraits**
  
  Wrapper class for experimental PebbldMinimizer.

- **class PecosApproximation**
  
  Derived approximation class for global basis polynomials.

- **class ProbabilityTransformModel**
  
  Probability transformation specialization of RecastModel.

- **class ProblemDescDB**
  
  The database containing information parsed from the DAKOTA input file.

- **class ProcessApplicInterface**
  
  Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.

- **class ProcessHandleApplicInterface**
  
  Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.

- **class ProgramOptions**
  
  ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag()

- **struct partial prp_hash**
  
  wrapper to delegate to the ParamResponsePair hash_value function

- **struct partial prp_equality**
  
  predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

- **class PSUADEDesignCompExp**
  
  Wrapper class for the PSUADE library.

- **class PythonInterface**

- **class RandomFieldModel**

- **class RecastModel**
  
  Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

- **class ReducedBasis**

- **class RelaxedVarConstraints**
  
  Derived class within the Constraints hierarchy which employs relaxation of discrete variables.

- **class RelaxedVariables**
  
  Derived class within the Variables hierarchy which employs the relaxation of discrete variables.

- **class ResultsDBAny**

- **class ResultsID**
  
  Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each Iterator::run() call creates or increments this count for its string identifier.
• class ResultsNames
  List of valid names for iterator results.
• class ResultsManager
  Results manager for iterator final data.
• class ResultsEntry
  Class to manage in-core vs. file database lookups.
• class RichExtrapVerification
  Class for Richardson extrapolation for code and solution verification.
• class ScalingModel
  Scaling specialization of RecastModel.
• class ScilabInterface
• class SensAnalysisGlobal
  Class for a utility class containing correlation calculations and variance-based decomposition.
• class SeqHybridMetaIterator
  Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.
• class SharedApproxData
  Base class for the shared approximation data class hierarchy.
• class SharedPecosApproxData
  Derived approximation class for global basis polynomials.
• class SharedResponseDataRep
  The representation of a SharedResponseData instance. This representation, or body, may be shared by multiple SharedResponseData handle instances.
• class SharedResponseData
  Container class encapsulating variables data that can be shared among a set of Response instances.
• class SharedSurfpackApproxData
  Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.
• class SharedVariablesDataRep
  The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances.
• class SharedVariablesData
  Container class encapsulating variables data that can be shared among a set of Variables instances.
• class SimulationModel
  Derived model class which utilizes a simulation-based application interface to map variables into responses.
• class SimulationResponse
  Container class for response functions and their derivatives. SimulationResponse provides the body class.
• class SNLLBase
  Base class for OPT++ optimization and least squares methods.
• class SNLLLeastSqTraits
  Wrapper class for the OPT++ optimization library.
• class SNLLTraits
  Wrapper class for the OPT++ optimization library.
• class SOLBase
  Base class for Stanford SOL software.
12.1. **DAKOTA NAMESPACE REFERENCE**

- **class SpawnApplicInterface**
  
  Derived application interface class which spawns simulation codes using spawnvp.

- **class SurfpackApproximation**
  
  Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

- **class SurrBasedGlobalTraits**
  
  The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls.

- **class SurrBasedGlobalMinimizer**
- **class SurrBasedLocalMinimizer**
  
  Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

- **class SurrBasedMinimizer**
  
  Base class for local/global surrogate-based optimization/least squares.

- **class SurrogateModel**
  
  Base class for surrogate models (DataFitSurrModel and HiearchSurrModel).

- **class SysCallApplicInterface**
  
  Derived application interface class which spawns simulation codes using system calls.

- **class TANA3Approximation**
  
  Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation).

- **class TaylorApproximation**
  
  Derived approximation class for first- or second-order Taylor series (a local approximation).

- **class TestDriverInterface**
- **class TrackerHTTP**
  
  TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.

- **class UsageTracker**
  
  Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL.

- **class VPSApproximation**
  
  Derived approximation class for VPS implementation.

- **struct MatchesWC**
  
  Predicate that returns true when the passed path matches the wildcard with which it was configured. Currently supports * and ?.

- **class WorkdirHelper**

**Typedefs**

- typedef double **Real**
- typedef std::string **String**
- typedef Teuchos::SerialDenseVector< int, Real > **RealVector**
- typedef Teuchos::SerialDenseMatrix< int, Real > **RealMatrix**
- typedef Teuchos::SerialSymDenseMatrix< int, Real > **RealSymMatrix**
• typedef
  Teuchos::SerialDenseVector
  < int, int >\textbf{IntVector}
• typedef
  Teuchos::SerialDenseMatrix
  < int, int >\textbf{IntMatrix}
• typedef std::deque< bool >\textbf{BoolDeque}
• typedef boost::dynamic_bitset
  < unsigned long >\textbf{BitArray}
• typedef std::vector< BoolDeque >\textbf{BoolDequeArray}
• typedef std::vector< Real >\textbf{RealArray}
• typedef std::vector< RealArray >\textbf{Real2DArray}
• typedef std::vector< int >\textbf{IntArray}
• typedef std::vector< IntArray >\textbf{Int2DArray}
• typedef std::vector< short >\textbf{ShortArray}
• typedef std::vector< unsigned short >\textbf{UShortArray}
• typedef std::vector< UShortArray >\textbf{UShort2DArray}
• typedef std::vector< size_t >\textbf{SizetArray}
• typedef std::vector< SizetArray >\textbf{Sizet2DArray}
• typedef std::vector< String >\textbf{StringArray}
• typedef std::vector< StringArray >\textbf{String2DArray}
• typedef boost::multi_array_types::index_range\textbf{idx_range}
• typedef boost::multi_array
  < String, 1 >\textbf{StringMultiArray}
• typedef boost::multi_array
  < String, 2 >\textbf{StringMulti2DArray}
• typedef
  StringMultiArray::array_view
  < 1 >\textbf{type StringMultiArrayView}
• typedef
  StringMultiArray::const_array_view
  < 1 >\textbf{type StringMultiArrayConstView}
• typedef boost::multi_array
  < unsigned short, 1 >\textbf{UShortMultiArray}
• typedef
  UShortMultiArray::array_view
  < 1 >\textbf{type UShortMultiArrayView}
• typedef
  UShortMultiArray::const_array_view
  < 1 >\textbf{type UShortMultiArrayConstView}
• typedef boost::multi_array
  < size_t, 1 >\textbf{SizetMultiArray}
typedef SizetMultiArray::array_view < 1 >::type SizetMultiArrayView
typedef SizetMultiArray::const_array_view < 1 >::type SizetMultiArrayConstView
typedef boost::multi_array < Real, 1 > RealMultiArray
typedef boost::multi_array < Real, 2 > RealMulti2DArray
typedef boost::multi_array < Real, 3 > RealMulti3DArray
typedef std::vector < RealVector > RealVectorArray
typedef std::vector < RealVectorArray > RealVector2DArray
typedef std::vector < RealMatrix > RealMatrixArray
typedef std::vector < RealSymMatrix > RealSymMatrixArray
typedef std::vector < IntVector > IntVectorArray
typedef std::vector < Variables > VariablesArray
typedef std::vector < Response > ResponseArray
typedef std::vector < ParamResponsePair > PRPArray
typedef std::vector < PRPArray > PRP2DArray
typedef std::vector < Model > ModelArray
typedef std::vector < Iterator > IteratorArray
typedef std::vector < RealMultiArray > BoostMAArray
typedef std::vector < RealMulti2DArray > BoostMA2DArray
typedef std::vector < RealMulti3DArray > BoostMA3DArray
typedef std::list < bool > BoolList
typedef std::list < int > IntList
typedef std::list < size_t > SizetList
typedef std::list < Real > RealList
typedef std::list < RealVector > RealVectorList
typedef std::list < String > StringList
typedef std::list < Variables > VariablesList
typedef std::list < Interface > InterfaceList
typedef std::list < Response > ResponseList
typedef std::list < ParamResponsePair > PRPList
typedef std::list < Model > ModelList
typedef std::list < Iterator > IteratorList
typedef std::pair < int, int > IntIntPair
typedef std::pair < size_t, size_t > SizetSizetPair
typedef std::pair
< SizetSizetPair, SizetSizetPair > SizetSizet2DPair
typedef std::pair< size_t, int > SizetIntPair
typedef std::pair< int, size_t > IntSizetPair
typedef std::pair< int, String > IntStringPair
typedef std::pair< Real, Real > RealRealPair
typedef std::pair< int, Response > IntResponsePair
typedef std::set< Real > RealSet
typedef std::set< int > IntSet
typedef std::set< String > StringSet
typedef std::set< unsigned short > UShortSet
typedef std::set< size_t > SizetSet
typedef std::set< RealRealPair > RealRealPairSet
typedef std::vector< RealSet > RealSetArray
typedef std::vector< IntSet > IntSetArray
typedef std::vector< StringSet > StringSetArray
typedef std::vector< UShortSet > UShortSetArray
typedef std::map< int, int > IntIntMap
typedef std::map< int, short > IntShortMap
typedef std::map< int, Real > IntRealMap
typedef std::map< Real, Real > RealRealMap
typedef std::map< String, Real > StringRealMap
typedef std::multimap< Real, int > RealIntMultiMap
typedef std::vector< RealRealPair > RealRealPairArray
typedef std::vector< IntRealMap > IntRealMapArray
typedef std::vector< StringRealMap > StringRealMapArray
typedef std::map< int, RealVector > IntRealVectorMap
typedef std::map< int, RealMatrix > IntRealMatrixMap
typedef std::map< int, ActiveSet > IntActiveSetMap
typedef std::map< int, Variables > IntVariablesMap
typedef std::map< int, Response > IntResponseMap
typedef std::map< IntArray, size_t > IntArraySizetMap
typedef std::map< IntIntPair, Real > IntIntPairRealMap
typedef std::map< IntIntPair, RealMatrix > IntIntPairRealMatrixMap
typedef std::map< RealRealPair, Real > RealRealPairRealMap
typedef std::vector< IntIntPairRealMap > IntIntPairRealMapArray
typedef std::vector< RealRealPairRealMap > RealRealPairRealMapArray
• `typedef std::multimap
  
  < RealRealPair, ParamResponsePair > RealPairPRPMultiMap`

• `typedef std::map
  
  < SizetSizet2DPair, DiscrepancyCorrection > DiscrepCorrMap`

• `typedef IntList::iterator ILIter`

• `typedef IntList::const_iterator ILCIter`

• `typedef SizetList::iterator StLIter`

• `typedef SizetList::const_iterator StLCIter`

• `typedef RealList::iterator RLIter`

• `typedef RealList::const_iterator RLCIter`

• `typedef RealVectorList::iterator RVLIter`

• `typedef RealVectorList::const_iterator RVLCIter`

• `typedef StringList::iterator StringLIter`

• `typedef StringList::const_iterator StringLCIter`

• `typedef VariablesList::iterator VarsLIter`

• `typedef InterfaceList::iterator InterfLIter`

• `typedef ResponseList::iterator RespLIter`

• `typedef PRPList::iterator PRPLIter`

• `typedef ModelList::iterator ModelLIter`

• `typedef ModelList::reverse_iterator ModelLRevIter`

• `typedef std::list
  
  < ParallelLevel >::iterator ParLevLIter`

• `typedef std::list
  
  < ParallelConfiguration > ::iterator ParConfigLIter`

• `typedef IntSet::iterator ISIter`

• `typedef IntSet::const_iterator ISCIter`

• `typedef StringSet::iterator SSIter`

• `typedef StringSet::const_iterator SSCIter`

• `typedef RealSet::iterator RSIter`

• `typedef RealSet::const_iterator RSCIter`

• `typedef IntIntMap::iterator IntIntMIter`

• `typedef IntIntMap::const_iterator IntIntMCIter`

• `typedef IntShortMap::iterator IntShMIter`

• `typedef IntShortMap::const_iterator IntShMCIter`

• `typedef IntRealMap::iterator IRMIter`

• `typedef IntRealMap::const_iterator IRMCIter`

• `typedef StringRealMap::iterator SRMIter`

• `typedef StringRealMap::const_iterator SRMCIter`

• `typedef RealRealMap::iterator RRMIter`

• `typedef RealRealMap::const_iterator RRMCIter`

• `typedef IntIntPairRealMap::iterator IIPRMIter`
typedef IntIntPairRealMap::const_iterator IIPRMCIter
typedef RealRealPairSet::iterator RRPSIter
typedef RealRealPairSet::const_iterator RRPSCIter
typedef RealRealPairRealMap::iterator RRPRMIter
typedef RealRealPairRealMap::const_iterator RRPRMCIter
typedef IntRealVectorMap::iterator IntRVMIter
typedef IntRealVectorMap::const_iterator IntRVMCIter
typedef IntRealMatrixMap::iterator IntRMMIter
typedef IntRealMatrixMap::const_iterator IntRMMCIter
typedef IntActiveSetMap::iterator IntASMIter
typedef IntVariablesMap::iterator IntVarsMIter
typedef IntVariablesMap::const_iterator IntVarsMCIter
typedef IntResponseMap::iterator IntRespMIter
typedef IntResponseMap::const_iterator IntRespMCIter
typedef boost::tuple<std::string, std::string, size_t, std::string> ResultsKeyType
Data type for results key (instance name / id, unique run, label), where data_key is a valid colon-delimited string from ResultsNames tuple<method_name, method_id, execution_number, data_key>

typedef std::string MetaDataKeyType
Data type for metadata key.

typedef std::vector<std::string> MetaDataValueType
Data type for metadata value.

typedef std::map<MetaDataKeyType, MetaDataValueType> MetaDataType

A single MetaData entry is map<string, vector<string>> Example: pair( "Column labels", ["Mean", "Std Dev", "Skewness", "Kurtosis"] )

typedef boost::tuple<std::string, std::string, size_t, std::string> > StrStrSizet
Iterator unique ID: <method_name, method_id, exec_num>

typedef void(* dl_core_run_t)(void *, Optimizer1 *, char *)

typedef void(* dl_destructor_t)(void **)

typedef Teuchos::SerialDenseSolver < int, Real > RealSolver

typedef Teuchos::SerialSpdDenseSolver < int, Real > RealSpdSolver
• typedef int(* start_grid_computing_t )(char *analysis_driver_script, char *params_file, char *results_file)
  definition of start grid computing type (function pointer)
• typedef int(* perform_analysis_t )(char *iteration_num)
  definition of perform analysis type (function pointer)
• typedef int (*)( get_jobs_completed_t )()
  definition of get completed jobs type (function pointer)
• typedef int(* stop_grid_computing_t )()
  definition of stop grid computing type (function pointer)
• typedef int MPI_Comm
• typedef void * MPI_Request
• typedef unsigned char u_char
• typedef unsigned short u_short
• typedef unsigned int u_int
• typedef unsigned long u_long
• typedef unsigned long UL
• typedef void(* Calcrj )(int *n, int *p, Real *x, int *nf, Real *r, int *ui, void *ur, Vf vf)
• typedef void(* Vf )()
• typedef void(* DbCallbackFunctionPtr )(Dakota::ProblemDescDB *db, void *data_ptr)
• typedef boost::tuple< bfs::path, bfs::path, bfs::path > PathTriple
  Triplet of filesystem paths: e.g., params, results, workdir.
• typedef bmi::multi_index_container
  < Dakota::ParamResponsePair,
  bmi::indexed_by
  < bmi::ordered_non_unique
  < bmi::tag< ordered >
  , bmi::const_mem_fun
  < Dakota::ParamResponsePair,
  const IntStringPair
  &,&Dakota::ParamResponsePair::eval_interface_ids >
  >, bmi::hashed_non_unique
  < bmi::tag< hashed >
  , bmi::identity
  < Dakota::ParamResponsePair >
  , partial_prp_hash,
  partial_prp_equality > > > PRPMultiIndexCache
  Boost Multi-Index Container for globally caching ParamResponsePairs.
• typedef PRPMultiIndexCache PRPCache
• typedef PRPCache::index_iterator
  < ordered >::type PRPCacheOIter
• typedef PRPCache::index_const_iterator
  < ordered >::type PRPCacheOCIter
• typedef PRPCache::index_iterator  
  < hashed >::type PRPCacheHIter
• typedef PRPCache::index_const_iterator  
  < hashed >::type PRPCacheHCIter
• typedef PRPCacheOIter PRPCacheIter  
  default cache iterator <0>
• typedef PRPCacheOCIter PRPCacheCIter  
  default cache const iterator <0>
• typedef boost::reverse_iterator  
  < PRPCacheCIter > PRPCacheCRevIter
• typedef bmi::multi_index_container  
  < Dakota::ParamResponsePair,  
    bmi::indexed_by  
    < bmi::ordered_unique  
      < bmi::tag< ordered >  
        , bmi::const_mem_fun  
        < Dakota::ParamResponsePair,  
          int,&Dakota::ParamResponsePair::eval_id >  
        , bmi::hashed_non_unique  
        < bmi::tag< hashed >  
        , bmi::identity  
        < Dakota::ParamResponsePair >  
        , partial_prp_hash,  
        partial_prp_equality > > > PRPMultiIndexQueue  
  
  Boost Multi-Index Container for locally queueing ParamResponsePairs.
• typedef PRPMultiIndexQueue PRPQueue
• typedef PRPQueue::index_iterator  
  < ordered >::type PRPQueueOIter
• typedef PRPQueue::index_const_iterator  
  < ordered >::type PRPQueueOCIter
• typedef PRPQueue::index_iterator  
  < hashed >::type PRPQueueHIter
• typedef PRPQueue::index_const_iterator  
  < hashed >::type PRPQueueHCIter
• typedef PRPQueueOIter PRPQueueIter
• typedef PRPQueueOCIter PRPQueueCIter
• typedef std::pair< boost::any,  
  MetaDataType > ResultsValueType  

Core data storage type: boost::any, with optional metadata (see other types in results_types.hpp)
• typedef boost::function< bool(const
tfs::path &src_path, const
tfs::path &dest_path, bool
overwrite) > file_op_function

define a function type that operates from src to dest, with option to overwrite

• typedef boost::filter_iterator
< MatchesWC,
tfs::directory_iterator > glob_iterator

a glob_iterator filters a directory_iterator based on a wildcard predicate

Enumerations

• enum { COBYLA, DIRECT, EA, MS,
PS, SW, BETA }

• enum {
APPROX_ERROR = -8, METHOD_ERROR = -7, MODEL_ERROR = -6, IO_ERROR = -5,
INTERFACE_ERROR = -4, CONSTRUCT_ERROR = -3, PARSE_ERROR = -2, OTHER_ERROR
= -1 }
enum for Dakota abort reasons; using negative numbers to distinguish Dakota exit states from signals / uncaught
signals. These need to be in range [-63, -1], so exit code (256+enum) is in [193, 255]. See RATIONALE in
dakota_global_defs.cpp.

• enum { [ ABORT_EXITS, ABORT_THROWS ]
enum for dakota abort behaviors

• enum { CV_ID_DEFAULT = 0, MINIMUM_METRIC, RELATIVE_TOLERANCE, DECREASE_T-
TOLERANCE }
enum for active subspace cross validation identification

• enum {
TABULAR_NONE = 0, TABULAR_HEADER = 1, TABULAR_EVAL_ID = 2, TABULAR_IFACE_ID
= 4,
TABULAR_EXPERANNOT = TABULAR_HEADER | TABULAR_EVAL_ID, TABULARANNOTATED
= TABULAR_HEADER | TABULAR_EVAL_ID | TABULAR_IFACE_ID }
options for tabular columns

• enum { [ FLEXIBLE_RESULTS, LABELED_RESULTS ]
options for results file format

• enum {
NO_MODEL_FORMAT =0, TEXT_ARCHIVE =1, BINARY_ARCHIVE =2, ALGEBRAIC_FILE =4,
ALGEBRAIC_CONSOLE =8 }
define special values for surrogateExportFormats

• enum CONSTRAINT_TYPE { LINEAR, NONLINEAR }

• enum LINEAR_INEQUALITY_FORMAT { none, two_sided, one_sided_lower, one_s-
ided_upper }

• enum NONLINEAR_EQUALITY_FORMAT { none, tpl_managed, dakota_managed }

• enum NONLINEAR_INEQUALITY_FORMAT { none, two_sided, one_sided_lower, o-
ne_sided_upper }
CHAPTER 12. NAMESPACE DOCUMENTATION

• enum {
  DEFAULT_INTERFACE = 0, APPROX_INTERFACE, FORK_INTERFACE = PROCESS_INTERFACE_BIT, SYSTEM_INTERFACE, GRID_INTERFACE, TEST_INTERFACE = DIRECT_INTERFACE_BIT, MATLAB_INTERFACE, PYTHON_INTERFACE, SCILAB_INTERFACE
}

  special values for interface type

• enum { SYNCHRONOUS_INTERFACE, ASYNCHRONOUS_INTERFACE }

  interface synchronization types

• enum { OBJECTIVE, INEQUALITY_CONSTRAINT, EQUALITY_CONSTRAINT }

  define algebraic function types

• enum {
  DEFAULT_METHOD = 0, HYBRID = (META_BIT | PARALLEL_BIT), PARETO_SET, MULTISTART,
  RICHARDSON_EXTRAP = (ANALYZER_BIT | VERIF_BIT), CENTERED_PARAMETER_STUDY = (ANALYZER_BIT | PSTDYDACE_BIT), LIST_PARAMETER_STUDY, MULTIDIM_PARAMETER_STUDY,
  VECTOR_PARAMETER_STUDY, DACE, FSU_CVT, FSU_HALTON, FSU_HAMMERSLEY, PSADE_MOAT, LOCAL_RELIABILITY = (ANALYZER_BIT | NOND_BIT), GLOBAL_RELIABILITY,
  POLYNOMIAL_CHAOS, STOCH_ColLOCATION, MULTILEVEL_POLYNOMIAL_CHAOS, MULTIFIDELITY_POLYNOMIAL_CHAOS,
  MULTIFIDELITY_STOCH_ColLOCATION, CUBATURE_INTEGRATION, SPARSE_GRID_INTEGRATION, QUADRATURE_INTEGRATION,
  BAYES_CALIBRATION, GPAIS, POF_DARTS, RKD_DARTS,
  IMPORTANCE_SAMPLING, ADAPTIVE_SAMPLING, MULTILEVEL_SAMPLING, LIST_SAMPLING,
  RANDOM_SAMPLING, LOCAL_INTERVAL_EST, LOCAL_EVIDENCE, GLOBAL_INTERVAL_EST,
  GLOBAL_EVIDENCE, SURROGATE_BASED_LOCAL = (MINIMIZER_BIT | SURRBASED_BIT), DATA_FIT_SURROGATE_BASED_LOCAL, HIERR_SURROGATE_BASED_LOCAL,
  SURROGATE_BASED_GLOBAL, EFFICIENT_GLOBAL, NL2_SOL = (MINIMIZER_BIT | LEASTSQ_BIT), NLSSOL_SQP,
  OPTPP_G_NEWTON, ASYNCH_PATTERN_SEARCH = (MINIMIZER_BIT | OPTIMIZER_BIT), OPTPP_PDS, COLINY_BETA,
  COLINY_COBYLA, COLINY_DIRECT, COLINY_MULTI_START, COLINY_EA,
  COLINY_PATTERN_SEARCH, COLINY_SOLIS_WETS, MOGA, SOGA,
  NCSU_DIRECT, MESH_ADAPTIVE_SEARCH, MIT_NOWPAC, MIT_SNOWPAC,
  GENIE_OPT_DARTS, GENIE_DIRECT, NONLINEAR_IMG, OPTPP_IMG,
  OPTPP_Q_NEWTON, OPTPP_FD_NEWTON, OPTPP_NEWTON, NPSOL_SQP,
  NLPQL_SQP, DOT_BFGS, DOT_FRCG, DOT_MMFD,
  DOT_SLP, DOT_SQP, CONMIN_FRCG, CONMIN_MFD,
  DL_SOLVER, BRANCH_AND_BOUND = (MINIMIZER_BIT | OPTIMIZER_BIT | LEASTSQ_BIT) }

• enum {
  SUBMETHOD_DEFAULT = 0, SUBMETHOD_NONE, SUBMETHOD_COLLABORATIVE, SUBMETHOD_EMBEDDED,
  SUBMETHODSEQUENTIAL, SUBMETHOD_LHS, SUBMETHOD_RANDOM, SUBMETHOD_BOX_BEHNKEN,
  SUBMETHODCENTRAL_COMPOSITE, SUBMETHOD_GRID, SUBMETHOD_OA_LHS, SUB-
METHOD_OAS, SUBMETHOD_DREAM, SUBMETHOD_GPMSA, SUBMETHOD_QUESO, SUBMETHOD_WASABI, SUBMETHOD_NIP, SUBMETHOD_SQP, SUBMETHOD_EA, SUBMETHOD_EGO, SUBMETHOD_SBO, SUBMETHOD_CONVERGE_ORDER, SUBMETHOD_CONVERGE_QOI, SUBMETHOD_ESTIMATE_ORDER

Sub-methods, including sampling, inference algorithm, opt algorithm types.

• enum { SILENT_OUTPUT, QUIET_OUTPUT, NORMAL_OUTPUT, VERBOSE_OUTPUT, DEBUG_OUTPUT } 
• enum { NO_RESULTS, INTERMEDIATE_RESULTS, FINAL_RESULTS } 
• enum { DEFAULT_SCHEDULING, MASTER_SCHEDULING, PEER_SCHEDULING, PEER_DYNAMIC_SCHEDULING, PEER_STATIC_SCHEDULING, DYNAMIC_SCHEDULING, STATIC_SCHEDULING } 
• enum { DEFAULT_CONFIG, PUSH_DOWN, PUSH_UP } 
• enum { STD_NORMAL_U, STD_UNIFORM_U, ASKEY_U, EXTENDED_U } 
• enum { DEFAULT_COVARIANCE, NO_COVARIANCE, DIAGONAL_COVARIANCE, FULL_COVARIANCE } 
• enum { NO_INT_REFINE =0, IS, AIS, MAIS } 
• enum { PROBABILITIES, RELIABILITIES, GEN_RELIABILITIES } 
• enum { COMPONENT =0, SYSTEM_SERIES, SYSTEM_PARALLEL } 
• enum { CUMULATIVE, COMPLEMENTARY } 
• enum { NO_MOMENTS, STANDARD_MOMENTS, CENTRAL_MOMENTS } 
• enum { NO_EMULATION, DISTINCT_EMULATION, RECURSIVE_EMULATION } 
• enum { DEFAULT_LS =0, SVD_LS, EQ_CON_LS } 
• enum { NO_EMULATOR, PCE_EMULATOR, ML_PCE_EMULATOR, MF_PCE_EMULATOR, SC_EMULATOR, MF_SC_EMULATOR, GP_EMULATOR, KRIGING_EMULATOR, VPS_EMULATOR } 
• enum { CALIBRATE_NONE = 0, CALIBRATE.ONE, CALIBRATE_PER_EXPER, CALIBRATE_PER_R-ESP, CALIBRATE_BOTH } 
• enum { IGNORE_RANKS, SET_RANKS, GET_RANKS, SET_GET_RANKS } 
• enum { UNCERTAIN, UNCERTAIN_UNIFORM, ALEATORY_UNCERTAIN, ALEATORY_UNCERTAIN_UNIFORM, EPISTEMIC_UNCERTAIN, EPISTEMIC_UNCERTAIN_UNIFORM, ACTIVE, ACTIVE_UNIFORM, ALL, ALL_UNIFORM } 
• enum { ONE_SIDED_LOWER, ONE_SIDEDLOWER, ONE_SIDED_LOWER, ONE_SIDED_UPPER, ONE_SIDED_UPPER, ONE_SIDED_UPPER, TWO_SIDED, TWO_SIDED } 
• enum { MV =0, AMV_X, AMV_U, AMV_PLUS_X, AMV_PLUS_U, TANA_X, TANA_U, NO_APPROX, EGRA_X, EGRA_U }
• enum { BREITUNG, HOHENRACK, HONG }
• enum { ORIGINAL_PRIMARY, SINGLE_OBJECTIVE, LAGRANGIAN_OBJECTIVE, AUGMENTED_LAGRANGIAN_OBJECTIVE }
• enum { NO_CONSTRAINTS, LINEARIZED_CONSTRAINTS, ORIGINAL_CONSTRAINTS }
• enum { NO_RELAX, HOMOTOPY, COMPOSITE_STEP }
• enum { PENALTY_MERIT, ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, AUGMENTED_LAGRANGIAN_MERIT }
• enum { FILTER, TR_RATIO }
• enum { SCALE_NONE, SCALE_VALUE, SCALE_LOG }
• enum { CDV, LINEAR, LINEAR, NONLIN, FN_LSQ }
• enum { DISALLOW, TARGET, BOUNDS }
• enum { DEFAULT_POINTS, MINIMUM_POINTS, RECOMMENDED_POINTS, TOTAL_POINTS }

  define special values for pointsManagement

• enum { NO_SURROGATE=0, UNCORRECTED_SURROGATE, AUTO_CORRECTED_SURROGATE, BYPASS_SURROGATE, MODEL_DISCREPANCY, AGGREGATED_MODELS }

  define special values for SurrogateModel::responseMode

• enum { NO_CORRECTION=0, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, COMBINED_CORRECTION }

  define special values for approxCorrectionType

• enum { RF_KARHUNEN_LOEVE=0, RF_PCA_GP, RF_ICA }

  define types of random field approximations

• enum { NOCOVAR=0, EXP_L2, EXP_L1 }

  define types of analytic covariance functions

• enum { SUBSPACE_NORM_DEFAULT=0, SUBSPACE_NORM_MEAN_VALUE, SUBSPACE_NORM_MEAN_GRAD, SUBSPACE_NORM_LOCAL_GRAD }

  define special values for active subspace normalizations

• enum { SURROGATE_MODEL=1, TRUTH_MODEL }

  define special values for componentParallelMode (active model for parallel scheduling)

• enum { BASE_RESPONSE=0, SIMULATION_RESPONSE, EXPERIMENT_RESPONSE }

  special values for derived Response type

• enum { GENERIC_FNS = 0, OBJECTIVE_FNS, CALIB_TERMS }

  values for primary response types

• enum { DEFAULT_DOMAIN=0, RELAXED_DOMAIN, MIXED_DOMAIN }

• enum { DEFAULT_VIEW =0, ALL_VIEW, DESIGN_VIEW, UNCERTAIN_VIEW, ALEATORY_UNCERTAIN_VIEW, EPISTEMIC_UNCERTAIN_VIEW, STATE_VIEW }

• enum { EMPTY_VIEW =0, RELAXED_ALL, MIXED_ALL, RELAXED_DESIGN, RELAXED_UNCERTAIN, RELAXED_ALEATORY_UNCERTAIN, RELAXED_EPISTEMIC_UNCERTAIN, RELAXED_STATE, MIXED_DESIGN, MIXED_UNCERTAIN, MIXED_ALEATORY_UNCERTAIN, MIXED_EPISTEMIC_UNCERTAIN, MIXED_STATE }
• enum \{ ALL_VARS = 0, ACTIVE_VARS, INACTIVE_VARS \} 

values differentiating subsets of variables for I/O

• enum 

EMPTY_TYPE = 0, CONTINUOUS_DESIGN, DISCRETE_DESIGN_RANGE, DISCRETE_DESIGN_SET_INT, DISCRETE_DESIGN_SET_STRING, DISCRETE_DESIGN_SET_REAL, NORMAL_UNCERTAIN, LOGNORMAL_UNCERTAIN, UNIFORM_UNCERTAIN, LOGUNIFORM_UNCERTAIN, TRIANGULAR_UNCERTAIN, EXPONENTIAL_UNCERTAIN, BETA_UNCERTAIN, GAMMA_UNCERTAIN, GUMBEL_UNCERTAIN, FRECHET_UNCERTAIN, WEIBULL_UNCERTAIN, HISTOGRAM_BIN_UNCERTAIN, POISSON_UNCERTAIN, BINOMIAL_UNCERTAIN, NEGATIVE_BINOMIAL_UNCERTAIN, GEOMETRIC_UNCERTAIN, HYPERGEOMETRIC_UNCERTAIN, HISTOGRAM_POINT_UNCERTAIN_INT, HISTOGRAM_POINT_UNCERTAIN_STRING, HISTOGRAM_POINT_UNCERTAIN_REAL, CONTINUOUS_INTERVAL_UNCERTAIN, DISCRETE_INTERVAL_UNCERTAIN, DISCRETE_UNCERTAIN_SET_INT, DISCRETE_UNCERTAIN_SET_STRING, DISCRETE_UNCERTAIN_SET_REAL, CONTINUOUS_STATE, DISCRETE_STATE_RANGE, DISCRETE_STATE_SET_INT, DISCRETE_STATE_SET_STRING, DISCRETE_STATE_SET_REAL \}

• enum 

TOTAL_CDV = 0, TOTAL_DDIV, TOTAL_DDSV, TOTAL_DDRV, TOTAL_CAUV, TOTAL_DAUIV, TOTAL_DAUSV, TOTAL_DAUERV, TOTAL_CEUV, TOTAL_DEUIV, TOTAL_DEUSV, TOTAL_DEURV, TOTAL_CSV, TOTAL_DSIV, TOTAL_DSSV, TOTAL_DSRV, NUM_VC_TOTALS \}

• enum var_t 

VAR_x1, VAR_x2, VAR_x3, VAR_b, VAR_h, VAR_P, VAR_M, VAR_Y, VAR_w, VAR_t, VAR_R, VAR_E, VAR_X, VAR_Fs, VAR_P1, VAR_P2, VAR_P3, VAR_B, VAR_D, VAR_H, VAR_F0, VAR_d, VAR_MForm \}

enumeration of possible variable types (to index to names)

• enum driver_t 

NO_DRIVER = 0, CANTILEVER_BEAM, MOD_CANTILEVER_BEAM, CYLINDER_HEAD, EXTENDED_ROSENBROCK, GENERALIZED_ROSENBROCK, LF_ROSENBROCK, EXTRA_LF_ROSENBROCK, MF_ROSENBROCK, MODIFIED_ROSENBROCK, ROSENBROCK, LF_POLY_PROD, POLY_PROD, GERSTNER, SCALABLE_GERSTNER, LOGNORMAL_RATIO, MULTIMODAL, PLUGIN_ROSENBROCK, PLUGIN_TEXT_BOOK, SHORT_COLUMN, LF_SHORT_COLUMN, MF_SHORT_COLUMN, SIDE_IMPACT_COST, SIDE_IMPACT_PERFO-
CHAPTER 12. NAMESPACE DOCUMENTATION

RMANCE,
SOBOL_RATIONAL, SOBOL_G_FUNCTION, SOBOLISHIGAMI, STEELCOLUMN_COST,
STEELCOLUMN_PERFORMANCE, TEXT_BOOK, TEXT_BOOK1, TEXT_BOOK2,
TEXT_BOOK3, TEXT_BOOK_OUU, SCALABLE_TEXT_BOOK, SCALABLE_MONOMIALS,
MOGATEST1, MOGATEST2, MOGATEST3, ILLUMINATION,
BARNES, BARNES_LF, HERBIE, SMOOTHHERBIE,
SHUBERT, SALINAS, MODELCENTER, GENZ,
DAMPEDOSCILLATOR, ANISOTROPIC_QUADRATIC_FORM, BAYES_LINEAR, STEADYSTATE_DIFFUSION_1D,
TRANSIENT_DIFFUSION_ID, PREDATOR_PREY
}

• enum local_data_t { VARIABLES_MAP = 1, VARIABLES_VECTOR = 2 }

• enum sigtype { NO_SIGMA, SCALAR_SIGMA, DIAGONAL_SIGMA, MATRIX_SIGMA }

• enum edtype { SCALAR_DATA, FUNCTIONAL_DATA }

• enum { DEFAULT_CORRECTION = 0, SINGLE_CORRECTION, FULL_MODEL_FORM_CORRECTION,
FULL_SOLUTION_LEVEL_CORRECTION, SEQUENCE_CORRECTION } }

• enum { SETUP_MODEL, SETUP_USERFUNC }

• enum {
CAUVar_normal = 0, CAUVar_lognormal = 1, CAUVar_uniform = 2, CAUVar_loguniform = 3,
CAUVar_triangular = 4, CAUVar_exponential = 5, CAUVar_beta = 6, CAUVar_gamma = 7,
CAUVar_gumbel = 8, CAUVar_frechet = 9, CAUVar_weibull = 10, CAUVar_histogram_bin = 11,
CAUVar_Nkinds = 12 }

• enum {
DAUIVar_poisson = 0, DAUIVar_binomial = 1, DAUIVar_negative_binomial = 2, DAUIVar_geometric = 3,
DAUIVar_hypergeometric = 4, DAUIVar_histogram_point_int = 5, DAUIVar_Nkinds = 6 }

• enum { DAUSVar_histogram_point_str = 0, DAUSVar_Nkinds = 1 }

• enum { DAURVar_histogram_point_real = 0, DAURVar_Nkinds = 1 }

• enum { CEUVar_interval = 0, CEUVar_Nkinds = 1 }

• enum { DEUVar_interval = 0, DEUVar_set_int = 1, DEUVar_Nkinds = 2 }

• enum { DEUVar_set_str = 0, DEUVar_Nkinds = 1 }

• enum { DEUVar_set_real = 0, DEUVar_Nkinds = 1 }

• enum {
DiscSetVar_design_set_int = 0, DiscSetVar_design_set_str = 1, DiscSetVar_design_set_real = 2, DiscSetVar_state_set_int = 3,
DiscSetVar_state_set_str = 4, DiscSetVar_state_set_real = 5, DiscSetVar_Nkinds = 6 }

• enum { NUM_UNC_REAL_CONT = 4 }

• enum { NUM_UNC_INT_CONT = 2 }

• enum { NUM_UNC_STR_CONT = 2 }

• number of real-valued uncertain contiguous containers
• number of int-valued uncertain contiguous containers
• number of string-valued uncertain contiguous containers
12.1. DAKOTA NAMESPACE REFERENCE

- **enum** `{ FULL_TENSOR, FILTERED_TENSOR, RANDOM_TENSOR }`
- **enum** `CG_UPDATE_TYPE`
  - `CG_STEEPEST, CG_FLETCHER_REEVES, CG_POLAK_RIBIERE, CG_POLAK_RIBIERE_PULS, CG_HESTENES_STIEFEL }`
  *NonlinearCG update options.*
- **enum** `CG_LINESEARCHTYPE` `{ CG_FIXED_STEP, CG_LS_SIMPLE, CG_LS_BRENT, CG_LS_WOLFE }`
  *NonlinearCG linesearch options.*
- **enum** `EvalType` `{ NLFEvaluator, CONEvaluator }
  - *enumeration for the type of evaluator function*
- **enum** `{ APPROX_RESPONSE =1, TRUTH_RESPONSE }`
- **enum** `{ CORR_APPROX_RESPONSE =1, UNCORR_APPROX_RESPONSE, CORR_TRUTH_RESPONSE, UNCORR_TRUTH_RESPONSE }`
- **enum** `{ NEW_CANDIDATE = 1, CANDIDATE_ACCEPTED = 2, CANDIDATE_STATE = (NEW_CANDIDATE | CANDIDATE_ACCEPTED), NEW_CENTER = 8, CENTER_BUILT = 16, CENTER_STATE = (NEW_CENTER | CENTER_BUILT), NEW_TR_FACTOR = 64, NEW_TRUST_REGION = (NEW_CENTER | NEW_TR_FACTOR), HARD_CONVERGED = 128, SOFT_CONVERGED = 256, MIN_TR_CONVERGED = 512, MAX_ITER_CONVERGED = 1024, CONVERGED }`
- **enum** `{ TH_SILENT_OUTPUT, TH_QUIET_OUTPUT, TH_NORMAL_OUTPUT, TH_VERBOSE_OUTPUT, TH_DEBUG_OUTPUT }`
- **enum** `{ DIR_CLEAN, DIR_PERSIST, DIR_ERROR }`
  *define directory creation options*
- **enum** `{ FILEOP_SILENT, FILEOP_WARN, FILEOP_ERROR }`
  *enum indicating action on failed file operation*

**Functions**

- **CommandShell & flush (CommandShell &shell)**
  - *convenient shell manipulator function to "flush" the shell*
- **void read sized data (std::istream &s, RealVectorArray &va, size_t num_rows, int num_cols)**
- **void read fixed rowsize data (std::istream &s, RealVectorArray &va, int num_cols, bool row_major)**
- **void read unsized data (std::istream &s, RealVectorArray &va, bool row_major)**
- **void read config vars mult file (const std::string &basename, int num_expts, int ncv, RealVectorArray &config_vars)**
- **void read config vars single file (const std::string &basename, int num_expts, int ncv, RealVectorArray &config_vars)**
- **void read field values (const std::string &basename, int expt_num, RealVectorArray &field_vars)**
- **void read field values (const std::string &basename, int expt_num, RealVector &field_vars)**
- **void read coord values (const std::string &basename, int expt_num, RealMatrix &coords)**
- **void read coord values (const std::string &basename, RealMatrix &coords)**
- **void read covariance (const std::string &basename, int expt_num, RealMatrix &cov_vals)**

• **void read_covariance** (const std::string &basename, int expt_num, Dakota::CovarianceMatrix::FORMAT format, int num_vals, RealMatrix &cov_vals)

  *tolerance-based equality operator for RealVector*

• **bool nearby** (const RealVector &rv1, const RealVector &rv2, Real rel_tol)

  *equality operator for RealVector*

• **bool operator==** (const ShortArray &dsa1, const ShortArray &dsa2)

  *equality operator for ShortArray*

• **bool operator==** (const StringArray &dsa1, const StringArray &dsa2)

  *equality operator for StringArray*

• **Real rel_change_L2** (const RealVector &curr_rv, const RealVector &prev_rv)

  *Computes relative change between RealVectors using Euclidean L2 norm.*

• **Real rel_change_L2** (const RealVector &curr_rv1, const RealVector &prev_rv1, const IntVector &curr_iv, const IntVector &prev_iv, const RealVector &curr_rv2, const RealVector &prev_rv2)

  *Computes relative change between Real/int/Real vector triples using Euclidean L2 norm.*

• **void remove_column** (RealMatrix &matrix, int index)

  *Removes column from matrix.*

• **void iround** (const RealVector &input_vec, IntVector &rounded_vec)

  *round entries of a RealVector yielding an IntVector*

• **bool operator==** (const IntArray &dia1, const IntArray &dia2)

  *equality operator for IntArray*

• **template<typename T>**

  **bool operator==** (const std::vector<T> &vec, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)

  *equality operator for std::vector and boost::multi_array::const_array_view*

• **template<typename T>**

  **bool operator==** (typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const std::vector<T>& vec)

  *equality operator for boost::multi_array::const_array_view and std::vector*

• **template<typename T>**

  **bool operator==** (const boost::multi_array<T, 1>& ma, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)

  *equality operator for boost::multi_array and boost::multi_array::const_array_view*

• **template<typename T>**

  **bool operator==** (typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const boost::multi_array<T, 1>& ma)

  *equality operator for boost::multi_array::const_array_view and boost::multi_array*

• **bool operator!=** (const IntArray &dia1, const IntArray &dia2)

  *inequality operator for IntArray*

• **bool operator!=** (const ShortArray &dsa1, const ShortArray &dsa2)

  *inequality operator for ShortArray*

• **bool operator!=** (const StringArray &dsa1, const StringArray &dsa2)

  *inequality operator for StringArray*

• **template<typename T>**

  **bool operator!=** (const std::vector<T> &vec, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)

  *inequality operator for std::vector and boost::multi_array::const_array_view*
inequality operator for std::vector and boost::multi_array::const_array_view

- template<typename T>
  bool operator!=(typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const std::vector<T>& vec)

inequality operator for boost::multi_array::const_array_view and std::vector

- template<typename T>
  bool operator!=(const boost::multi_array<T, 1>& ma, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)

inequality operator for boost::multi_array and boost::multi_array::const_array_view

- template<typename T>
  bool operator!=(typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const boost::multi_array<T, 1>& ma)

std::string strtolower (const std::string &s)
Return lowercase copy of string s.

- bool strbegins (const std::string &input, const std::string &test)
Return true if input string begins with string test.

- bool strends (const std::string &input, const std::string &test)
Return true if input string ends with string test.

- bool strcontains (const std::string &input, const std::string &test)
Return true if input string contains string test.

void build_label (String &label, const String &root_label, size_t tag, const String &separator="")
create a label by appending a numerical tag to the root label, o

- void build_labels (StringArray &label_array, const String &root_label)
create an array of labels by tagging root_label for each entry in label_array. Uses build_label().

- void build_labels (StringMultiArray &label_array, const String &root_label)
create an array of labels by tagging root_label for each entry in label_array. Uses build_label().

- void build_labels_partial (StringArray &label_array, const String &root_label, size_t start_index, size_t num_items)
create a partial array of labels by tagging root_label for a subset of entries in label_array. Uses build_label().

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const std::vector<Teuchos::SerialDenseVector<OrdinalType, ScalarType>>& sda, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType>& sdm)
copy Array<Teuchos::SerialDenseVector<OT,ST>> to Teuchos::SerialDenseMatrix<OT,ST> - used by read_data_tabular - RWH

- template<typename OrdinalType, typename ScalarType>
  void copy_data_transpose (const std::vector<Teuchos::SerialDenseVector<OrdinalType, ScalarType>>& sda, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType>& sdm)
copy Array<Teuchos::SerialDenseVector<OT,ST>> to transposed Teuchos::SerialDenseMatrix<OT,ST> - used by read_data_tabular - RWH

- template<typename OrdinalType1, typename OrdinalType2, typename ScalarType>
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType1, ScalarType>& sv, Teuchos::SerialDenseMatrix<OrdinalType1, ScalarType>& sdm, OrdinalType2 nr, OrdinalType2 nc)
copy Teuchos::SerialDenseVector<OT,ST> to Teuchos::SerialDenseMatrix<OT,ST> - used by NestedModel::update_sub_iterator - RWH
• template<typename T>
  void copy_data (const std::vector< std::vector< T >> &d2a, std::vector< T > &da)
    copy std::vector< std::vector< T >> to std::vector< T > (unroll vecOfvecs into vector) - used by ProcessApplicInterface::write_parameters_files - RWH

• template<typename T>
  void copy_data (const std::map< int, T > &im, std::vector< T > &da)
    copy map<int, T> to std::vector<T> (discard integer keys) - used by SurrBasedGlobalMinimizer::core_run - RWH

• template<typename OrdinalType , typename ScalarType >
  void copy_data (const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &sdv1, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &sdv2)
    copy Teuchos::SerialDenseVector<OrdinalType, ScalarType> to same (used in place of operator= when a deep copy of a vector view is needed) - used by SensAnalysisGlobal::valid_sample_matrix - RWH

• template<typename OrdinalType1 , typename OrdinalType2 , typename ScalarType >
  void copy_data (const Teuchos::SerialDenseVector< OrdinalType1, ScalarType > &sdv, ScalarType *ptr, const OrdinalType2 ptr_len)
    copy portion of first SerialDenseVector to all of second SerialDenseVector - used by DataTransformModel::vars_mapping - RWH
• template<typename OrdinalType1 , typename OrdinalType2 , typename ScalarType >
  void copy_data_partial (const Teuchos::SerialDenseVector< OrdinalType1, ScalarType > &sdv1, Teuchos::SerialDenseVector< OrdinalType1, ScalarType > &sdv2, OrdinalType2 start_index2)
  
    copy all of first SerialDenseVector to portion of second SerialDenseVector - used by MixedVariables - RWH, NLS-SOLLeastSq - BMA

• template<typename OrdinalType1 , typename OrdinalType2 , typename ScalarType >
  void copy_data_partial (const Teuchos::SerialDenseVector< OrdinalType1, ScalarType > &sdv1, OrdinalType2 start_index1, OrdinalType2 num_items, Teuchos::SerialDenseVector< OrdinalType1, ScalarType > &sdv2, OrdinalType2 start_index2)
  
    copy portion of first SerialDenseVector to portion of second SerialDenseVector - used by ScalingModel::secondary_resps_scaled2native - RWH

• template<typename OrdinalType1 , typename OrdinalType2 , typename ScalarType >
  void copy_data_partial (const Teuchos::SerialDenseVector< OrdinalType1, ScalarType > &sdv1, std::vector<ScalarType> &da2, OrdinalType2 start_index2)
  
    copy all of first SerialDenseVector to portion of second SerialDenseVector - used by SharedSurfpackApproxData::merge_variable_arrays - RWH

• template<typename T >
  void copy_data_partial (const std::vector<T> &da1, size_t start_index1, size_t num_items, std::vector<T> &da2)
  
    copy portion of first Array<T> to all of second Array<T> - used by SharedResponseDataRep constructor - RWH

• template<typename T >
  void copy_data_partial (const std::vector<T> &da1, std::vector<T> &da2, size_t start_index2)
  
    copy all of first Array<T> to portion of second Array<T> - used by ParamStudy::multidim_loop - RWH

• template<typename T >
  void copy_data_partial (const std::vector<T> &da, boost::multi_array<T, 1> &bma, size_t start_index_bma)
  
    copy all of first Array<T> to portion of boost::multi_array<T, 1> - used by RelaxedVariables - RWH

• template<typename VectorType >
  void copy_row_vector (const RealMatrix &m, RealMatrix::ordinalType i, VectorType &row)
  
    Copies a row of a Teuchos_SerialDenseMatrix<int,Real> to std::vector<Real>

• template<typename ScalarType >
  void insert_row_vector (const std::vector<ScalarType> & &row, RealMatrix::ordinalType i, RealMatrix &m)
  
    Copies a row of a Teuchos_SerialDenseMatrix<int,Real> to std::vector<Real>

• void merge_data_partial (const IntVector &d_vec, RealVector &m_vec, size_t start_index_ma)
  
    merge a discrete integer vector into a single continuous vector

• void merge_data_partial (const IntVector &d_vec, RealArray &m_array, size_t start_index_ma)
  
    merge a discrete integer vector into a single continuous array

• template<typename OrdinalType , typename ScalarType >
  const ScalarType & set_index_to_value (OrdinalType index, const std::set<ScalarType> & &values)
  
    retrieve the set value corresponding to the passed index

• template<typename ScalarType >
  size_t set_value_to_index (const ScalarType &value, const std::set<ScalarType> & &values)
  
    calculate the set index corresponding to the passed value

• template<typename OrdinalType , typename KeyType , typename ValueType >
  const KeyType & map_index_to_key (OrdinalType index, const std::map<KeyType, ValueType> & &pairs)
CHAPTER 12. NAMESPACE DOCUMENTATION

retrieve the set value corresponding to the passed index
- template<typename OrdinalType, typename KeyType, typename ValueType>
  const ValueType & map_index_to_value (OrdinalType index, const std::map<KeyType, ValueType> &pairs)
  retrieve the set value corresponding to the passed index
- template<typename KeyType, typename ValueType>
  void map_keys_to_set (const std::map<KeyType, ValueType> &source_map, std::set<KeyType> &target_set)
    calculate the map index corresponding to the passed key
- template<typename KeyType, typename ValueType>
  size_t map_key_to_index (const KeyType &key, const std::map<KeyType, ValueType> &pairs)
    calculate the map index corresponding to the passed key
- template<typename KeyType, typename ValueType>
  size_t map_value_to_index (const ValueType &value, const std::map<KeyType, ValueType> &pairs)
    calculate the map index corresponding to the passed value
- template<typename OrdinalType, typename ScalarType>
  void x_y_pairs_to_x_set (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &xy_pairs, std::set<ScalarType> &x_set)
    convert a SerialDenseVector of head-to-tail (x,y) pairs into a std::set of (x), discarding the y values
- template<typename ContainerType>
  size_t find_index (const ContainerType &c, const typename ContainerType::value_type &search_data)
    generic find_index (inactive)
- template<typename T>
  size_t find_index (const boost::multi_array<T, 1> &bma, const T &search_data)
    compute the index of an entry within a boost::multi_array
- size_t find_index (SizetMultiArrayConstView bmacv, size_t search_data)
  compute the index of an entry within a boost::multi_array view
- size_t find_index (StringMultiArrayConstView bmacv, const String &search_data)
  compute the index of an entry within a boost::multi_array view
- template<typename ListT>
  size_t find_index (const ListT &l, const typename ListT::value_type &val)
    compute the index of an entry within a std::list
- template<typename ListT>
  ListT::const_iterator find_if (const ListT &c, bool(*test_fn)(const typename ListT::value_type &, const std::string &), const std::string &test_fn_data)
    return an iterator to the first list element satisfying the predicate test_fn w.r.t. the passed test_fn_data; end if not found
- template<typename VectorType, typename ScalarType>
  void copy_data (const std::vector<VectorType> &va, ScalarType *ptr, int ptr_len)
- void copy_data (SizetMultiArrayConstView ma, SizetArray &da)
  copy boost::multi_array view to Array - used by ActiveSet::derivative_vector - RWH
- void copy_data (StringMultiArrayConstView ma, StringArray &da)
  copy boost::multi_array view to Array - used by Pecos::copy_data - RWH
- template<typename DakContainerType>
  bool contains (const DakContainerType &v, const typename DakContainerType::value_type &val)
    return true if the item val appears in container v
12.1. DAKOTA NAMESPACE REFERENCE

- void abort_handler (int code)
  
  global function which handles serial or parallel aborts
- void abort_throw_or_exit (int dakota_code)
  
  throw or exit depending on abort_mode
- void register_signal_handlers ()
  
  Tie various signal handlers to Dakota's abort_handler function.
- void mpi_debug_hold ()
  
  Global function to hold Dakota processes to help with MPI debugging.
- template<
type>
  T abort_handler_t (int code)
- void svd (RealMatrix &matrix, RealVector &singular_vals, RealMatrix &v_trans, bool compute_vectors=true)
  
  Compute the SVD of an arbitrary matrix $A = USV^T$.
- void singular_values (RealMatrix &matrix, RealVector &singular_values)
  
  compute the singular values without storing any singular vectors ($A$ will be destroyed)
- int qr (RealMatrix &A)
  
  Compute an in-place QR factorization $A = QR$.
- int qr_rsolve (const RealMatrix &q_r, bool transpose, RealMatrix &rhs)
  
  Perform a multiple right-hand sides $Rinv \times rhs$ solve using the $R$ from a qr factorization.
- double det_AtransA (RealMatrix &A)
  
  Use SVD to compute det$(A^\top A)$, destroying $A$ with the SVD.
- ResultsKeyType make_key (const StrStrSizet &iterator_id, const std::string &data_name)
  
  Make a full ResultsKeyType from the passed iterator_id and data_name.
- MetaDataValueType make_metadatavalue (StringMultiArrayConstView labels)
  
  create MetaDataValueType from the passed strings
- MetaDataValueType make_metadatavalue (StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels)
  
  create MetaDataValueType from the passed strings
- MetaDataValueType make_metadatavalue (const StringArray &resp_labels)
  
  create MetaDataValueType from the passed strings
- MetaDataValueType make_metadatavalue (const std::string &)
  
  create MetaDataValueType from the passed strings
- MetaDataValueType make_metadatavalue (const std::string &)
  
  create MetaDataValueType from the passed strings
- MetaDataValueType make_metadatavalue (const std::string &)
  
  create MetaDataValueType from the passed strings
- MetaDataValueType make_metadatavalue (const std::string &)
  
  create MetaDataValueType from the passed strings
- MetaDataValueType make_metadatavalue (StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView dsv_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels)
  
- std::istream & operator>>(std::istream &s, ActiveSet &set)
  
  std::istream extraction operator for ActiveSet. Calls read(std::istream&).
- std::ostream & operator<<(std::ostream &s, const ActiveSet &set)
std::ostream insertion operator for ActiveSet. Calls write(std::istream&).

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ActiveSet &set)

MPIUnpackBuffer extraction operator for ActiveSet. Calls read(MPIUnpackBuffer&).

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const ActiveSet &set)

MPIPackBuffer insertion operator for ActiveSet. Calls write(MPIPackBuffer&).

- bool operator!=(const ActiveSet &set1, const ActiveSet &set2)

inequality operator for ActiveSet

- std::istream & operator>>(std::istream &s, Constraints &con)

std::istream extraction operator for Constraints

- std::ostream & operator<<(std::ostream &s, const Constraints &con)

std::ostream insertion operator for Constraints

- std::string re_match(const std::string &token, const boost::regex &re)

Global utility function to ease migration from CtelRegExp to Boost.Regex.

- bool interface_id_compare(const Interface &interface_in, const void *id)

global comparison function for Interface

- bool method_id_compare(const Iterator &iterator, const void *id)

global comparison function for Iterator

- bool model_id_compare(const Model &model, const void *id)

global comparison function for Model

- bool operator==(const Model &m1, const Model &m2)

equality operator for Envelope is true if same letter instance

- bool operator!=(const Model &m1, const Model &m2)

inequality operator for Envelope is true if different letter instance

- template<typename VecT>
  bool get_bounds(const RealVector &lower_source, const RealVector &upper_source, Real big_real_bound_size, Real no_value, VecT &lower_target, VecT &upper_target)

- template<typename SetT, typename VecT>
  void get_bounds(const SetT &source_set, VecT &lower_target, VecT &upper_target, int target_offset)

- template<typename OrdinalType, typename ScalarType, typename VectorType2, typename MaskType, typename SetArray>
  bool get_mixed_bounds(const MaskType &mask_set, const SetArray &source_set, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &lower_source, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &upper_source, VectorType2 &lower_target, VectorType2 &upper_target, ScalarType bigBoundSize, ScalarType no_value, int target_offset=0)

- template<typename AdapterT>
  bool get_variable_bounds(Model &model, Real real_bound_size, int int_bound_size, typename AdapterT::VecT &lower, typename AdapterT::VecT &upper)

- template<typename RVecT, typename IVecT>
  int configure_inequality_constraint_maps(const Model &model, Real real_bound_size, CONSTRAINT_TYPE ctype, IVecT &map_indices, RVecT &map_multipliers, RVecT &map_offsets, Real scaling=1.0)

- template<typename RVecT, typename IVecT>
  void configure_inequality_constraint_maps(Model &model, CONSTRAINT_TYPE ctype, IVecT &indices, size_t index_offset, RVecT &multipliers, RVecT &values, bool make_one_sided)

- template<typename AdapterT>
  void get_linear_constraints(Model &model, Real real_bound_size, typename AdapterT::VecT &lin_ineq_lower_bnds, typename AdapterT::VecT &lin_ineq_upper_bnds, typename AdapterT::VecT &lin_eq_targets, typename AdapterT::MatT &lin_ineq_coeffs, typename AdapterT::MatT &lin_eq_coeffs)
12.1. DAKOTA NAMESPACE REFERENCE

- template<
typename VectorType1 , typename VectorType2 , typename SetArray >
  void copy_data (const VectorType1 &source, const BitArray &set_bits, const SetArray &set_vars, VectorType2 &dest, size_t offset, size_t len)
- template<
typename VectorType1 , typename VectorType2 , typename SetArray >
  void copy_data (const VectorType1 &source, const BitArray &set_bits, const SetArray &set_vars, VectorType2 &dest, size_t offset, size_t len)
- template<
typename VectorType1 , typename VectorType2 >
  void copy_data (const VectorType1 &source, const SetArray &set_vars, VectorType2 &dest, size_t offset, size_t len)
- template<
typename VectorType1 , typename VectorType2 , typename IntSetArray >
  void copy_data (const VectorType1 &source, const IntSetArray &set_int_vars, size_t offset, size_t len)
- template<
typename AdapterT>
  void set_best_responses (typename AdapterT::OptT &optimizer, const Model &model, const std::vector<int> constraint_map_indices, const std::vector<double> constraint_map_multipliers, const std::vector<double> constraint_map_offsets, ResponseArray &response_array)
- template<
typename VectorType >
  void set_variables (const VectorType &source, Model &model, Variables &vars)
- template<
typename VectorType >
  void get_variables (Model &model, VectorType &vec)
- template<
typename VectorType >
  void get_responses (const Model &model, const RealVector &dak_fn_vals, const std::vector<int> constraint_map_indices, const std::vector<double> constraint_map_multipliers, const std::vector<double> constraint_map_offsets, VectorType &f_vec, VectorType &cEqs_vec, VectorType &cIneqs_vec)
- template<
typename RVecT >
  void get_nonlinear_eq_constraints (Model &model, const RVecT &curr_resp_vals, RVecT &values)
- template<
typename RVecT >
  void get_nonlinear_eq_constraints (Model &model, const RealVector &curr_resp_vals, RVecT &values, Real scale, int offset=0)
- template<
typename VecT >
  void get_nonlinear_constraints (Model &model, VecT &nonlin_ineq_lower, VecT &nonlin_ineq_upper, VecT &nonlin_eq_targets)

  Would like to combine the previous adapter with this one (based on APPSOptimizer and COLINOptimizer) and then see how much more generalization is needed to support other TPLs like JEGA.

- bool responses_id_compare (const Response &resp, const void *id)
  global comparison function for Response

  std::istream & operator>>(std::istream &s, Response &response)
  std::istream extraction operator for Response. Calls read(std::istream&).

  std::ostream & operator<<(std::ostream &s, const Response &response)
  std::ostream insertion operator for Response. Calls write(std::ostream&).

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Response &response)
  MPIUnpackBuffer extraction operator for Response. Calls read(MPIUnpackBuffer&).

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const Response &response)
  MPIPackBuffer insertion operator for Response. Calls write(MPIPackBuffer&).

- bool operator!=(const Response &resp1, const Response &resp2)
  inequality operator for Response

- bool variables_id_compare (const Variables &vars, const void *id)
  global comparison function for Variables

- std::istream & operator>>(std::istream &s, Variables &vars)
std::ostream extraction operator for Variables.

- std::ostream & operator<<(std::ostream &s, const Variables &vars)

std::ostream insertion operator for Variables.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Variables &vars)

MPIUnpackBuffer extraction operator for Variables.

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const Variables &vars)

MPIPackBuffer insertion operator for Variables.

- bool operator!=(const Variables &vars1, const Variables &vars2)

inequality operator for Variables

- template<typename OrdinalType, typename ScalarType1, typename ScalarType2, typename ScalarType3, typename ScalarType4>

  void write_ordered(std::ostream &s, const SizetArray &comp_totals, const Teuchos::SerialDenseVector<OrdinalType, ScalarType1> &c_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType2> &di_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType3> &ds_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType4> &dr_vector)

  free function to write Variables data vectors in input spec ordering

- template<typename OrdinalType, typename ScalarType1, typename ScalarType2, typename ScalarType3, typename ScalarType4>

  void write_ordered(std::ostream &s, const SizetArray &comp_totals, const Teuchos::SerialDenseVector<OrdinalType, ScalarType1> &c_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType2> &di_vector, const boost::multi_array<ScalarType3, 1> &ds_array, const Teuchos::SerialDenseVector<OrdinalType, ScalarType4> &dr_vector)

  free function to write Variables data vectors in input spec ordering

- template<typename ScalarType>

  void write_ordered(std::ostream &s, const SizetArray &comp_totals, const std::vector<ScalarType> &c_array, const std::vector<ScalarType> &di_array, const std::vector<ScalarType> &ds_array, const std::vector<ScalarType> &dr_array)

  free function to write Variables data vectors in input spec ordering

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataEnvironment &data)

MPIPackBuffer insertion operator for DataEnvironment.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataEnvironment &data)

MPIUnpackBuffer extraction operator for DataEnvironment.

- std::ostream & operator<<(std::ostream &s, const DataEnvironment &data)

std::ostream insertion operator for DataEnvironment

- String interface_enum_to_string (unsigned short interface_type)

MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataInterface &data)

MPIPackBuffer insertion operator for DataInterface.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataInterface &data)

MPIUnpackBuffer extraction operator for DataInterface.

- std::ostream & operator<<(std::ostream &s, const DataInterface &data)

std::ostream insertion operator for DataInterface

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataMethod &data)

MPIPackBuffer insertion operator for DataMethod.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataMethod &data)

MPIUnpackBuffer extraction operator for DataMethod.
12.1. DAKOTA NAMESPACE REFERENCE

- std::ostream & operator<<(std::ostream &s, const DataMethod &data)
  std::ostream insertion operator for DataMethod.
- MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataModel &data)
  MPIPackBuffer insertion operator for DataModel.
- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataModel &data)
  MPIUnpackBuffer extraction operator for DataModel.
- std::ostream & operator<<(std::ostream &s, const DataModel &data)
  std::ostream insertion operator for DataModel.
- MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataResponses &data)
  MPIPackBuffer insertion operator for DataResponses.
- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataResponses &data)
  MPIUnpackBuffer extraction operator for DataResponses.
- std::ostream & operator<<(std::ostream &s, const DataResponses &data)
  std::ostream insertion operator for DataResponses.
- MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataVariables &data)
  MPIPackBuffer insertion operator for DataVariables.
- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataVariables &data)
  MPIUnpackBuffer extraction operator for DataVariables.
- std::ostream & operator<<(std::ostream &s, const DataVariables &data)
  std::ostream insertion operator for DataVariables.
- int dl_solver_option (Opt_Info *)
- RealVector const * continuous_lower_bounds (Optimizer1 *o)
- RealVector const * continuous_upper_bounds (Optimizer1 *o)
- RealVector const * nonlinear_ineq_constraint_lower_bounds (Optimizer1 *o)
- RealVector const * nonlinear_ineq_constraint_upper_bounds (Optimizer1 *o)
- RealVector const * nonlinear_eq_constraint_targets (Optimizer1 *o)
- RealVector const * linear_ineq_constraint_lower_bounds (Optimizer1 *o)
- RealVector const * linear_ineq_constraint_upper_bounds (Optimizer1 *o)
- RealVector const * linear_eq_constraint_targets (Optimizer1 *o)
- RealMatrix const * linear_ineq_constraint_coeffs (Optimizer1 *o)
- RealMatrix const * linear_eq_constraint_coeffs (Optimizer1 *o)
- void ComputeResponses (Optimizer1 *o, int mode, int n, double *x)
- void GetFuncs (Optimizer1 *o, int m0, int m1, int n, int is, double *f)
- void GetGrads (Optimizer1 *o, int m0, int m1, int n, int is, int js, double *g)
- void GetContVars (Optimizer1 *o, int n, double *x)
- void SetBestContVars (Optimizer1 *o, int n, double *x)
- void SetBestRespFns (Optimizer1 *o, int n, double *x)
- void * dl_constructor (Optimizer1 *, Dakota_funcs *, dl_core_run_t *, dlDestructor_t *)
- static RealVector const * continuous_lower_bounds1 (Optimizer1 *o)
- static RealVector const * continuous_upper_bounds1 (Optimizer1 *o)
- static RealVector const * nonlinear_ineq_constraint_lower_bounds1 (Optimizer1 *o)
- static RealVector const * nonlinear_ineq_constraint_upper_bounds1 (Optimizer1 *o)
- static RealVector const * nonlinear_eq_constraint_targets1 (Optimizer1 *o)
- static RealVector const * linear_ineq_constraint_lower_bounds1 (Optimizer1 *o)
- static RealVector const * linear_ineq_constraint_upper_bounds1 (Optimizer1 *o)
• static RealVector const *\texttt{linear\_eq\_constraint\_targets1} (Optimizer1 *o)
• static RealMatrix const *\texttt{linear\_eq\_constraint\_coeffs1} (Optimizer1 *o)
• static RealMatrix const *\texttt{linear\_ineq\_constraint\_coeffs1} (Optimizer1 *o)
• static void \texttt{ComputeResponses1} (Optimizer1 *o, int mode, int n, double *x)
• static void \texttt{GetFuncs1} (Optimizer1 *o, int m0, int m1, double *f)
• static void \texttt{GetGrads1} (Optimizer1 *o, int m0, int m1, int n, int is, int js, double *g)
• static void \texttt{GetContVars1} (Optimizer1 *o, int n, double *x)
• static void \texttt{SetBestContVars1} (Optimizer1 *o, int n, double *x)
• static void \texttt{SetBestDiscVars1} (Optimizer1 *o, int n, int *x)
• static void \texttt{SetBestRespFns1} (Optimizer1 *o, int n, double *x)
• static double \texttt{Get\_Real1} (Optimizer1 *o, const char *name)
• static int \texttt{Get\_Int1} (Optimizer1 *o, const char *name)
• static bool \texttt{Get\_Bool1} (Optimizer1 *o, const char *name)
• \texttt{DOTOptimizer *new\_DOTOptimizer} (ProblemDescDB &problem, Model &model)
• \texttt{DOptimizer *new\_DOptimizer} (ProblemDescDB &problem, Model &model)
• void \texttt{copy\_field\_data} (const RealVector &fn_vals, RealMatrix &fn_grad, const RealSymMatrixArray &fn_hess, size_t offset, size_t num_fns, Response &response)
• void \texttt{copy\_field\_data} (const RealVector &fn_vals, RealMatrix &fn_grad, const RealSymMatrixArray &fn_hess, size_t offset, size_t num_fns, short total_asv, Response &response)
• void \texttt{interpolate\_simulation\_field\_data} (const Response &sim_resp, const RealMatrix &exp_coords, size_t field_num, short total_asv, size_t interp_resp_offset, Response &interp_resp)
• void \texttt{linear\_interpolate\_1d} (const RealMatrix &build_pts, const RealVector &build_vals, const RealMatrix &build_grads, const RealSymMatrixArray &build_hessians, const RealMatrix &pred_pts, RealVector &pred_vals, RealMatrix &pred_grads, RealSymMatrixArray &pred_hessians)

  Returns the value of at 1D function \(f\) and its gradient and hessians (if available) at the points of vector \(\text{pred}_\text{pts}\) using linear interpolation. The vector \(\text{build}_\text{pts}\) specifies the coordinates of the underlying interval at which the values (\(\text{build}_\text{vals}\)) of the function \(f\) are known. The length of output \(\text{pred}_\text{vals}\) is equal to the length of \(\text{pred}_\text{pts}\). This function assumes the \(\text{build}_\text{pts}\) is in ascending order.
• void \texttt{symmetric\_eigenvalue\_decomposition} (const RealSymMatrix &matrix, RealVector &eigenvalues, RealMatrix &eigenvectors)

  Computes the eigenvalues and, optionally, eigenvectors of a real symmetric matrix \(A\).
• void \texttt{compute\_column\_means} (RealMatrix &matrix, RealVector &avg_vals)

  Compute the means of each column of an arbitrary matrix.
• bool \texttt{is\_matrix\_symmetric} (const RealMatrix &matrix)

  Test if incoming matrix is symmetric.
• template<typename O , typename T >
  int \texttt{binary\_search} (T target, Teuchos::SerialDenseVector\langle O, T \rangle &data)

  find the interval containing a target value. This function assumes the data is in ascending order.
• Real \texttt{getdist} (const RealVector &x1, const RealVector &x2)
• Real \texttt{mindist} (const RealVector &x, const RealMatrix &xset, int except)
• Real \texttt{mindistindx} (const RealVector &x, const RealMatrix &xset, const IntArray &indx)
• Real \texttt{getRmax} (const RealMatrix &xset)
• int \texttt{start\_grid\_computing} (char *analysis\_driver\_script, char *params\_file, char *results\_file)
• int \texttt{stop\_grid\_computing} ()
• int \texttt{perform\_analysis} (char *iteration\_num)
12.1. DAKOTA NAMESPACE REFERENCE

- template<typename T>
  
  string asstring (const T &val)

  *Creates a string from the argument val using an ostringstream.*

- **PACKBUF** (int, MPI_INT) PACKBUF(u_int)
- **MPI_UNSIGNED** **PACKBUF** (long, MPI_LONG) PACKBUF(u_long)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG** **PACKBUF** (short, MPI_SHORT) PACKBUF(u_short)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG**
  
  MPI_UNSIGNED_SHORT **PACKBUF** (char, MPI_CHAR) PACKBUF(u_char)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG**
  
  MPI_UNSIGNED_SHORT **PACKBUF** (double, MPI_DOUBLE) PACKBUF(float)

- **UNPACKBUF** (int, MPI_INT) UNPACKBUF(u_int)
- **MPI_UNSIGNED** **UNPACKBUF** (long, MPI_LONG) UNPACKBUF(u_long)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG** **UNPACKBUF** (short, MPI_SHORT) UNPACKBUF(u_short)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG**
  
  MPI_UNSIGNED_SHORT **UNPACKBUF** (char, MPI_CHAR) UNPACKBUF(u_char)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG**
  
  MPI_UNSIGNED_SHORT **UNPACKBUF** (double, MPI_DOUBLE) UNPACKBUF(float)

- **PACKSIZE** (int, MPI_INT) PACKSIZE(u_int)
- **MPI_UNSIGNED** **PACKSIZE** (long, MPI_LONG) PACKSIZE(u_long)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG** **PACKSIZE** (short, MPI_SHORT) PACKSIZE(u_short)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG**
  
  MPI_UNSIGNED_SHORT **PACKSIZE** (char, MPI_CHAR) PACKSIZE(u_char)
- **MPI_UNSIGNED** **MPI_UNSIGNED_LONG**
  
  MPI_UNSIGNED_SHORT **PACKSIZE** (double, MPI_DOUBLE) PACKSIZE(float)

  int MPIPackSize (const bool &data, const int num=1)

  *return packed size of a bool*

  MPIPackBuffer & operator<< (MPIPackBuffer &buff, const int &data)

  *insert an int*

  MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_int &data)

  *insert a u_int*

  MPIPackBuffer & operator<< (MPIPackBuffer &buff, const long &data)

  *insert a long*

  MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_long &data)

  *insert a u_long*

  MPIPackBuffer & operator<< (MPIPackBuffer &buff, const short &data)

  *insert a short*

  MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_short &data)

  *insert a u_short*

  MPIPackBuffer & operator<< (MPIPackBuffer &buff, const char &data)

  *insert a char*
CHAPTER 12. NAMESPACE DOCUMENTATION

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_char &data)`
  insert a u_char
- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const double &data)`
  insert a double
- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const float &data)`
  insert a float
- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const bool &data)`
  insert a bool
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, int &data)`
  extract an int
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_int &data)`
  extract a u_int
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, long &data)`
  extract a long
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_long &data)`
  extract a u_long
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, short &data)`
  extract a short
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_short &data)`
  extract a u_short
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, char &data)`
  extract a char
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_char &data)`
  extract a u_char
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, double &data)`
  extract a double
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, float &data)`
  extract a float
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, bool &data)`
  extract a bool

- `template<
class ContainerT>
void container_read (ContainerT &c, MPIUnpackBuffer &s)`
  Read a generic container (vector<T>, list<T>) from MPIUnpackBuffer, s.
- `template<class ContainerT>
void container_write (const ContainerT &c, MPIPackBuffer &s)`
  Write a generic container to MPIPackBuffer, s.
- `template<
typename Block , typename Allocator>
MPIPackBuffer & operator<< (MPIPackBuffer &s, const boost::dynamic_bitset<Block, Allocator> &bs)`
  stream insertion for BitArray
- `template<
typename Block , typename Allocator>
MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, boost::dynamic_bitset<Block, Allocator> &bs)`
  stream extraction for BitArray
12.1. DAKOTA NAMESPACE REFERENCE

- template<class ContainerT>
  
  MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ContainerT &data)
  
  global MPIUnpackBuffer extraction operator for generic container

- template<class ContainerT>
  
  MPIPackBuffer & operator<<(MPIPackBuffer &s, const ContainerT &data)
  
  global MPIPackBuffer insertion operator for generic container

- int MPIPackSize (const int &data, const int num=1)
  
  return packed size of an int

- int MPIPackSize (const u_int &data, const int num=1)
  
  return packed size of a u_int

- int MPIPackSize (const long &data, const int num=1)
  
  return packed size of a long

- int MPIPackSize (const u_long &data, const int num=1)
  
  return packed size of a u_long

- int MPIPackSize (const short &data, const int num=1)
  
  return packed size of a short

- int MPIPackSize (const u_short &data, const int num=1)
  
  return packed size of a u_short

- int MPIPackSize (const char &data, const int num=1)
  
  return packed size of a char

- int MPIPackSize (const u_char &data, const int num=1)
  
  return packed size of a u_char

- int MPIPackSize (const double &data, const int num=1)
  
  return packed size of a double

- int MPIPackSize (const float &data, const int num=1)
  
  return packed size of a float

- int nidr_parse (const char *, FILE *)
  
  const char ** arg_list_adjust (const char **, void **)

- int not_executable (const char *driver_name, const char *tdir)

- static void scale_chk (StringArray &ST, RealVector &S, const char *what, const char **univ)

- static void BuildLabels (StringArray *sa, size_t nsa, size_t n1, size_t n2, const char *stub)

- static int mixed_check (IntSet *S, int n, IntArray *iv, const char *what)

- static int wronglen (size_t n, RealVector *V, const char *what)

- static int wronglen (size_t n, IntVector *V, const char *what)

- static void Vcopyup (RealVector *V, RealVector *M, size_t i, size_t n)

- static void Set_rv (RealVector *V, double d, size_t n)

- static void Set_iv (IntVector *V, int d, size_t n)

- static void wrong_number (const char *what, const char *kind, size_t nsv, size_t m)

- static void too_small (const char *kind)

- static void not_div (const char *kind, size_t nsv, size_t m)

- static void suppressed (const char *kind, int ndup, int *ip, String *sp, Real *rp)

- static void bad_initial_value (const char *kind, int val)

- static void bad_initial_value (const char *kind, String val)
• static void **bad_initial_rvalue** (const char *kind, Real val)
• static void **Vgen_ContinuousDes** (DataVariablesRep *dv, size_t offset)
• static void **Vgen_DiscreteDesRange** (DataVariablesRep *dv, size_t offset)
• static void **Vgen_DiscreteState** (DataVariablesRep *dv, size_t offset)
• static void **Vgen_DiscreteStateRange** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_NormalUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_NormalUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_LognormalUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_LognormalUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_UniformUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_UniformUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_LoguniformUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vgen_LoguniformUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_TriangularUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_TriangularUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_ExponentialUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_ExponentialUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_BetaUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_BetaUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_GammaUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_GammaUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_GumbelUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_GumbelUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_FrechetUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_FrechetUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_WeibullUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_WeibullUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_HistogramBinUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)

    Check the histogram bin input data, normalize the counts and populate the histogramUncBinPairs map data structure; map keys are guaranteed unique since the abscissas must increase.
• static void **Vgen_HistogramBinUnc** (DataVariablesRep *dv, size_t offset)

    Infer lower/upper bounds for histogram and set initial variable values based on initial_point or moments, snapping to bounds as needed. (Histogram bin doesn’t have lower/upper bounds specification)
• static void **Vchk_PoissonUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_PoissonUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_BinomialUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_BinomialUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_NegBinomialUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_NegBinomialUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_GeometricUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_GeometricUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_HyperGeomUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void **Vgen_HyperGeomUnc** (DataVariablesRep *dv, size_t offset)
• static void **Vchk_HistogramPointIntUnc** (DataVariablesRep *dv, size_t offset, Var_Info *vi)

    Check the histogram point integer input data, normalize the counts, and populate DataVariables::histogramUncPointIntPairs; map keys are guaranteed unique since the abscissas must increase.
• static void Vgen_HistogramPtIntUnc (DataVariablesRep *dv, size_t offset)
  
  Use the integer-valued point histogram data to initialize the lower, upper, and initial values of the variables, using
  value closest to mean if no initial point.

• static void Vchk_HistogramPtStrUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
  
  Check the histogram point string input data, normalize the counts, and populate DataVariables::histogramUnc-
  PointStrPairs; map keys are guaranteed unique since the abscissas must increase (lexicographically)

• static void Vgen_HistogramPtStrUnc (DataVariablesRep *dv, size_t offset)
  
  Use the string-valued point histogram data to initialize the lower, upper, and initial values of the variables, using
  index closest to mean index if no initial point.

• static void Vchk_HistogramPtRealUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
  
  Check the histogram point integer real data, normalize the counts, and populate DataVariables::histogramUnc-
  PointRealPairs; map keys are guaranteed unique since the abscissas must increase.

• static void Vgen_HistogramPtRealUnc (DataVariablesRep *dv, size_t offset)
  
  Use the real-valued point histogram data to initialize the lower, upper, and initial values of the variables, using
  value closest to mean if no initial point.

• static void Vchk_ContinuousIntervalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
  
  Check the continuous interval uncertain input data and populate DataVariables::continuousIntervalUncBasic-
  Probs; map keys (real intervals) are checked for uniqueness because we don’t have a theoretically sound way
to combine duplicate intervals.

• static void Vgen_ContinuousIntervalUnc (DataVariablesRep *dv, size_t offset)

• static void Vchk_DiscreteIntervalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
  
  Check the discrete interval uncertain input data and populate DataVariables::discreteIntervalUncBasicProbs;
  map keys (integer intervals) are checked for uniqueness because we don’t have a theoretically sound way
to combine duplicate intervals.

• static void Vchk_DIsset (size_t num_v, const char *kind, IntArray *input_ndsi, IntVector *input_dsi, IntSet-
  Array &dsi_all, IntVector &dsi_init_pt)

  check discrete sets of integers (design and state variables); error if a duplicate value is specified error if not ordered
to prevent user confusion

• static void Vchk_DIsset (size_t num_v, const char *kind, IntArray *input_ndsi, IntVector *input_dsi, Real-
  Vector *input_dsip, IntRealMapArray &dsi_vals_probs, IntVector &dsi_init_pt)

  check discrete sets of integers (uncertain variables); error if a duplicate value is specified error if not ordered to
  prevent user confusion

• static void Vchk_DSset (size_t num_v, const char *kind, IntArray *input_ndss, StringArray *input_dss,
  StringSetArray &dss_all, StringArray &dss_init_pt)

• static void Vchk_DSset (size_t num_v, const char *kind, IntArray *input_ndss, StringArray *input_dss,
  RealVector *input_dsip, RealRealMapArray &dss_vals_probs, StringArray &dss_init_pt)

• static void Vchk_DRset (size_t num_v, const char *kind, IntArray *input_nldr, RealVector *input_dsr,
  RealSetArray &dsr_all, RealVector &dsr_init_pt)

• static void Vchk_DRset (size_t num_v, const char *kind, IntArray *input_nldr, RealVector *input_dsr, Real-
  Vector *input_dsrp, RealRealMapArray &dsr_vals_probs, RealVector &dsr_init_pt)

• static void Vchk_Adjacency (size_t num_v, const char *kind, const IntArray &num_e, const IntVector
  &input_ddsa, RealMatrixArray &dda_all)
• static bool check_LUV_size (size_t num_v, IntVector &L, IntVector &U, IntVector &V, bool aggregate_LUV, size_t offset)

• static bool check_LUV_size (size_t num_v, StringArray &L, StringArray &U, StringArray &V, bool aggregate_LUV, size_t offset)

• static bool check_LUV_size (size_t num_v, RealVector &L, RealVector &U, RealVector &V, bool aggregate_LUV, size_t offset)

• static void Vgen_DSset (size_t num_v, IntSetArray &sets, IntVector &L, IntVector &U, IntVector &V, bool aggregate_LUV=false, size_t offset=0)

• static void Vgen_DSset (size_t num_v, StringSetArray &sets, StringArray &L, StringArray &U, StringArray &V, bool aggregate_LUV=false, size_t offset=0)

  generate lower, upper, and initial point for string-valued sets

• static void Vgen_DSset (size_t num_v, IntRealMapArray &vals_probs, IntVector &IP, IntVector &L, IntVector &U, IntVector &V, bool aggregate_LUV=false, size_t offset=0)

• static void Vgen_DRSset (size_t num_v, RealSetArray &sets, RealVector &L, RealVector &U, RealVector &V, bool aggregate_LUV=false, size_t offset=0)

• static void Vgen_DRSset (size_t num_v, RealRealMapArray &vals_probs, RealVector &IP, RealVector &L, RealVector &U, RealVector &V, bool aggregate_LUV=false, size_t offset=0)

• static void Vgen_DSset (size_t num_v, StringRealMapArray &vals_probs, StringVector &IP, StringVector &L, StringVector &U, StringVector &V, bool aggregate_LUV=false, size_t offset=0)

For real-valued variables: verify lengths of bounds and initial point, validate bounds and adjust initial point to bounds.

• static void Var_IntBoundIPCheck (DataVariablesRep *dv, Var_Info *vi)

  For integer-valued variables: verify lengths of bounds and initial point, validate bounds and initial point against bounds.

• static void flatten_rva (RealVectorArray *rva, RealVector **prv)

• static void flatten_iva (IntVectorArray *iva, IntVector **piv)

• static void flatten_rsm (RealSymMatrix *rsm, RealVector **prv)
static void flatten_rsa (RealSetArray *rsa, RealVector **prv)
static void flatten_ssa (StringSetArray *ssa, StringArray **psa)
static void flatten_isa (IntSetArray *isa, IntVector **piv)
static void flatten_rrma_keys (RealRealMapArray *rrma, RealVector **prv)
static void flatten_rrma_values (RealRealMapArray *rrma, RealVector **prv)
static void flatten_irma_keys (IntRealMapArray *irma, IntVector **piv)
static void flatten_irma_values (IntRealMapArray *irma, RealVector **prv)
static void flatten_sroma_keys (StringRealMapArray *sroma, StringArray **psa)
static void flatten_sroma_values (StringRealMapArray *sroma, RealVector **prv)
static void flatten_real_intervals (const RealRealPairRealMapArray &rrprma, RealVector **probs, RealVector **lb, RealVector **ub)
static void flatten_int_intervals (const IntIntPairRealMapArray &iiprma, RealVector **probs, IntVector **lb, IntVector **ub)

Flatten real-valued interval uncertain variable intervals and probabilities back into separate arrays.

static void var_iulbl (const char *keyname, Values *val, VarLabel *vl)
static Iface mp_Rlit MP3 (failAction, recoveryFnVals, recover)
static Iface mp_ilit MP3 (failAction, retryLimit, retry)
static Iface mp_ilit MP2 (failAction, abort)
static Iface mp_type MP2s (analysisScheduling, MASTER_SCHEDULING)
static Iface mp_type MP2s (analysisScheduling, PEER_SCHEDULING)
static Iface mp_type MP2s (evalScheduling, MASTER_SCHEDULING)
static Iface mp_type MP2s (evalScheduling, PEER_DYNAMIC_SCHEDULING)
static Iface mp_type MP2s (evalScheduling, PEER_STATIC_SCHEDULING)
static Iface mp_type MP2s (asyncLocalEvalScheduling, DYNAMIC_SCHEDULING)
static Iface mp_type MP2s (asyncLocalEvalScheduling, STATIC_SCHEDULING)
static Iface mp_type MP2s (interfaceSynchronization, ASYNCHRONOUS_INTERFACE)
static Iface mp_type MP2s (interfaceSynchronization, SYNCHRONOUS_INTERFACE)
static Iface mp_utype MP2s (interfaceType, TEST_INTERFACE)
static Iface mp_utype MP2s (interfaceType, FORK_INTERFACE)
static Iface mp_utype MP2s (interfaceType, GRID_INTERFACE)
static Iface mp_utype MP2s (interfaceType, MATLAB_INTERFACE)
static Iface mp_utype MP2s (interfaceType, PYTHON_INTERFACE)
static Iface mp_utype MP2s (interfaceType, SCILAB_INTERFACE)
static Iface mp_utype MP2s (interfaceType, SYSTEM_INTERFACE)
static Iface mp_utype MP2s (resultsFileFormat, LABELED_RESULTS)
static String MP_ (algebraicMappings)
static String MP_ (idInterface)
static String MP_ (inputFilter)
static String MP_ (outputFilter)
static String MP_ (parametersFile)
static String MP_ (resultsFile)
static String MP_ (workDir)
static String2DArray MP_ (analysisComponents)
static StringArray MP_ (analysisDrivers)
- static StringArray MP_ (copyFiles)
- static StringArray MP_ (linkFiles)
- static bool MP_ (activeSetVectorFlag)
- static bool MP_ (allowExistingResultsFlag)
- static bool MP_ (apreproFlag)
- static bool MP_ (dirSave)
- static bool MP_ (dirTag)
- static bool MP_ (evalCacheFlag)
- static bool MP_ (fileSaveFlag)
- static bool MP_ (fileTagFlag)
- static bool MP_ (nearbyEvalCacheFlag)
- static bool MP_ (numpyFlag)
- static bool MP_ (restartFileFlag)
- static bool MP_ (templateReplace)
- static bool MP_ (useWorkdir)
- static bool MP_ (verbatimFlag)
- static int MP_ (analysisServers)
- static int MP_ (asynchLocalAnalysisConcurrency)
- static int MP_ (asynchLocalEvalConcurrency)
- static int MP_ (evalServers)
- static int MP_ (procsPerAnalysis)
- static int MP_ (procsPerEval)
- static Real MP_ (nearbyEvalCacheTol)
- static IntVector MP_ (primeBase)
- static IntVector MP_ (refineSamples)
- static IntVector MP_ (sequenceLeap)
- static IntVector MP_ (sequenceStart)
- static IntVector MP_ (stepsPerVariable)
- static Method mp_lit2 MP3 (replacementType, numberRetained, chc)
- static Method mp_lit2 MP3 (replacementType, numberRetained, elitist)
- static Method mp_lit2 MP3 (replacementType, numberRetained, random)
- static Method mp_lit2z MP3 (crossoverType, numCrossPoints, multi_point_binary)
- static Method mp_lit2z MP3 (crossoverType, numCrossPoints, multi_point_parameterized_binary)
- static Method mp_lit MP2 (batchSelectionType, naive)
- static Method mp_lit MP2 (batchSelectionType, distance_penalty)
- static Method mp_lit MP2 (batchSelectionType, topology)
- static Method mp_lit MP2 (batchSelectionType, constant_liar)
- static Method mp_lit MP2 (boxDivision, all_dimensions)
- static Method mp_lit MP2 (boxDivision, major_dimension)
- static Method mp_lit MP2 (convergenceType, average_fitness_tracker)
- static Method mp_lit MP2 (convergenceType, best_fitness_tracker)
- static Method mp_lit MP2 (convergenceType, metric_tracker)
- static Method mp_lit MP2 (crossoverType, blend)
- static Method mp_lit MP2 (crossoverType, two_point)
- static Method mp_lit MP2 (crossoverType, uniform)
• static Method mp_lit MP2 (dataDistCovInputType, diagonal)
• static Method mp_lit MP2 (dataDistCovInputType, matrix)
• static Method mp_lit MP2 (evalSynchronize, blocking)
• static Method mp_lit MP2 (evalSynchronize, nonblocking)
• static Method mp_lit MP2 (expansionSampleType, incremental_lhs)
• static Method mp_lit MP2 (exploratoryMoves, adaptive)
• static Method mp_lit MP2 (exploratoryMoves, multi_step)
• static Method mp_lit MP2 (exploratoryMoves, simple)
• static Method mp_lit MP2 (fitnessType, domination_count)
• static Method mp_lit MP2 (fitnessType, layer_rank)
• static Method mp_lit MP2 (fitnessType, linear_rank)
• static Method mp_lit MP2 (fitnessType, merit_function)
• static Method mp_lit MP2 (fitnessType, proportional)
• static Method mp_lit MP2 (fitnessMetricType, predicted_variance)
• static Method mp_lit MP2 (fitnessMetricType, distance)
• static Method mp_lit MP2 (fitnessMetricType, gradient)
• static Method mp_lit MP2 (initializationType, random)
• static Method mp_lit MP2 (initializationType, unique_random)
• static Method mp_lit MP2 (lipschitzType, global)
• static Method mp_lit MP2 (lipschitzType, local)
• static Method mp_lit MP2 (meritFunction, merit_max)
• static Method mp_lit MP2 (meritFunction, merit_max_smooth)
• static Method mp_lit MP2 (meritFunction, merit1)
• static Method mp_lit MP2 (meritFunction, merit1_smooth)
• static Method mp_lit MP2 (meritFunction, merit2)
• static Method mp_lit MP2 (meritFunction, merit2_smooth)
• static Method mp_lit MP2 (mcmcType, adaptive_metropolis)
• static Method mp_lit MP2 (mcmcType, delayed_rejection)
• static Method mp_lit MP2 (mcmcType, dram)
• static Method mp_lit MP2 (mcmcType, metropolis_hastings)
• static Method mp_lit MP2 (mcmcType, multilevel)
• static Method mp_lit MP2 (modelDiscrepancyType, global_kriging)
• static Method mp_lit MP2 (modelDiscrepancyType, global_polynomial)
• static Method mp_lit MP2 (mutationType, bit_random)
• static Method mp_lit MP2 (mutationType, offset_cauchy)
• static Method mp_lit MP2 (mutationType, offset_normal)
• static Method mp_lit MP2 (mutationType, offset_uniform)
• static Method mp_lit MP2 (mutationType, replace_uniform)
• static Method mp_lit MP2 (patternBasis, coordinate)
• static Method mp_lit MP2 (patternBasis, simplex)
• static Method mp_lit MP2 (pointReuse, all)
• static Method mp_lit MP2 (proposalCovInputType, diagonal)
• static Method mp_lit MP2 (proposalCovInputType, matrix)
• static Method mp_lit MP2 (proposalCovType, derivatives)
• static Method mp_lit MP2 (proposalCovType, prior)
- static Method\_mp\_lit MP2 (proposalCovType, user)
- static Method\_mp\_lit MP2 (reliabilityIntegration, first\_order)
- static Method\_mp\_lit MP2 (reliabilityIntegration, second\_order)
- static Method\_mp\_lit MP2 (replacementType, elitist)
- static Method\_mp\_lit MP2 (replacementType, favor\_feasible)
- static Method\_mp\_lit MP2 (replacementType, roulette\_wheel)
- static Method\_mp\_lit MP2 (replacementType, unique\_roulette\_wheel)
- static Method\_mp\_lit MP2 (rngName, mt19937)
- static Method\_mp\_lit MP2 (rngName, rnum2)
- static Method\_mp\_lit MP2 (searchMethod, gradient\_based\_line\_search)
- static Method\_mp\_lit MP2 (searchMethod, tr\_pds)
- static Method\_mp\_lit MP2 (searchMethod, trust\_region)
- static Method\_mp\_lit MP2 (searchMethod, value\_based\_line\_search)
- static Method\_mp\_lit MP2 (trialType, grid)
- static Method\_mp\_lit MP2 (trialType, halton)
- static Method\_mp\_lit MP2 (trialType, random)
- static Method\_mp\_lit MP2 (useSurrogate, inform\_search)
- static Method\_mp\_lit MP2 (useSurrogate, optimize)
- static Method\_mp\_litc MP3 (crossoverType, crossoverRate, shuffle\_random)
- static Method\_mp\_litc MP3 (crossoverType, crossoverRate, null\_crossover)
- static Method\_mp\_litc MP3 (mutationType, mutationRate, null\_mutation)
- static Method\_mp\_litc MP3 (mutationType, mutationRate, offset\_cauchy)
- static Method\_mp\_litc MP3 (mutationType, mutationRate, offset\_normal)
- static Method\_mp\_litc MP3 (mutationType, mutationRate, offset\_uniform)
- static Method\_mp\_litc MP3 (replacementType, fitnessLimit, below\_limit)
- static Method\_mp\_litrv MP3 (nichingType, nicheVector, distance)
- static Method\_mp\_litrv MP3 (nichingType, nicheVector, max\_designs)
- static Method\_mp\_litrv MP3 (nichingType, nicheVector, radial)
- static Method\_mp\_litrv MP3 (nichingType, nicheVector, distance\_postprocessor)
- static Method\_mp\_slit2 MP3 (initializationType, flatFile, flat\_file)
- static Method\_mp\_utype\_lit MP3s (methodName, dlDetails, DL\_SOLVER)
- static Method\_mp\_ord MP2s (approx\_CorrectionOrder, 0)
- static Method\_mp\_ord MP2s (approx\_CorrectionOrder, 1)
- static Method\_mp\_ord MP2s (approx\_CorrectionOrder, 2)
- static Real MP\_ (absConvTol)
- static Real MP\_ (centeringParam)
- static Real MP\_ (collocationRatio)
- static Real MP\_ (collocRatioTerms\_Order)
- static Real MP\_ (constraintPenalty)
- static Real MP\_ (constrPenalty)
- static Real MP\_ (constraintTolerance)
- static Real MP\_ (contractFactor)
- static Real MP\_ (contractStep\_Length)
- static Real MP\_ (convergence\_Tolerance)
- static Real MP\_ (crossover\_Rate)
- static Real MP\_ (falseConv\_Tol)
• static Real \( \text{MP}_\text{c} \) (functionPrecision)
• static Real \( \text{MP}_\text{c} \) (globalBalanceParam)
• static Real \( \text{MP}_\text{c} \) (gradientTolerance)
• static Real \( \text{MP}_\text{c} \) (hybridLSProb)
• static Real \( \text{MP}_\text{c} \) (grThreshold)
• static Real \( \text{MP}_\text{c} \) (initDelta)
• static Real \( \text{MP}_\text{c} \) (initMeshSize)
• static Real \( \text{MP}_\text{c} \) (initStepLength)
• static Real \( \text{MP}_\text{c} \) (initTRRadius)
• static Real \( \text{MP}_\text{c} \) (lineSearchTolerance)
• static Real \( \text{MP}_\text{c} \) (localBalanceParam)
• static Real \( \text{MP}_\text{c} \) (maxBoxSize)
• static Real \( \text{MP}_\text{c} \) (maxStep)
• static Real \( \text{MP}_\text{c} \) (minBoxSize)
• static Real \( \text{MP}_\text{c} \) (minMeshSize)
• static Real \( \text{MP}_\text{c} \) (multilevEstimatorRate)
• static Real \( \text{MP}_\text{c} \) (mutationRate)
• static Real \( \text{MP}_\text{c} \) (mutationScale)
• static Real \( \text{MP}_\text{c} \) (percentVarianceExplained)
• static Real \( \text{MP}_\text{c} \) (refinementRate)
• static Real \( \text{MP}_\text{c} \) (regressionL2Penalty)
• static Real \( \text{MP}_\text{c} \) (shrinkagePercent)
• static Real \( \text{MP}_\text{c} \) (singConvTol)
• static Real \( \text{MP}_\text{c} \) (singRadius)
• static Real \( \text{MP}_\text{c} \) (smoothFactor)
• static Real \( \text{MP}_\text{c} \) (solnTarget)
• static Real \( \text{MP}_\text{c} \) (stepLenToBoundary)
• static Real \( \text{MP}_\text{c} \) (threshDelta)
• static Real \( \text{MP}_\text{c} \) (threshStepLength)
• static Real \( \text{MP}_\text{c} \) (trustRegionContract)
• static Real \( \text{MP}_\text{c} \) (trustRegionContractTrigger)
• static Real \( \text{MP}_\text{c} \) (trustRegionExpand)
• static Real \( \text{MP}_\text{c} \) (trustRegionExpandTrigger)
• static Real \( \text{MP}_\text{c} \) (trustRegionMinSize)
• static Real \( \text{MP}_\text{c} \) (vbdDropTolerance)
• static Real \( \text{MP}_\text{c} \) (volBoxSize)
• static Real \( \text{vns} \)
• static Real \( \text{MP}_\text{c} \) (wilksConfidenceLevel)
• static Real \( \text{MP}_\text{c} \) (xConvTol)
• static RealVector \( \text{MP}_\text{c} \) (anisoDimPref)
• static RealVector \( \text{MP}_\text{c} \) (concurrentParameterSets)
• static RealVector \( \text{MP}_\text{c} \) (dataDistCovariance)
• static RealVector \( \text{MP}_\text{c} \) (dataDistMeans)
• static RealVector \( \text{MP}_\text{c} \) (finalPoint)
• static RealVector \( \text{MP}_\text{c} \) (hyperPriorAlphas)
• static RealVector \( \text{MP}_\text{c} \) (hyperPriorBetas)
- static RealVector **MP_**(listOfPoints)
- static RealVector **MP_**(predictionConfigList)
- static RealVector **MP_**(proposalCovData)
- static RealVector **MP_**(regressionNoiseTol)
- static RealVector **MP_**(stepVector)
- static RealVector **MP_**(trustRegionInitSize)
- static RealVectorArray **MP_**(genReliabilityLevels)
- static RealVectorArray **MP_**(probabilityLevels)
- static RealVectorArray **MP_**(reliabilityLevels)
- static RealVectorArray **MP_**(responseLevels)
- static unsigned short **MP_**(adaptedBasisAdvancements)
- static unsigned short **MP_**(cubIntOrder)
- static unsigned short **MP_**(expansionOrder)
- static unsigned short **MP_**(quadratureOrder)
- static unsigned short **MP_**(softConvLimit)
- static unsigned short **MP_**(sparseGridLevel)
- static unsigned short **MP_**(vbdOrder)
- static unsigned short **MP_**(wilksOrder)
- static SizetArray **MP_**(collocationPointsSeq)
- static SizetArray **MP_**(expansionSamplesSeq)
- static SizetArray **MP_**(pilotSamples)
- static UShortArray **MP_**(expansionOrderSeq)
- static UShortArray **MP_**(quadratureOrderSeq)
- static UShortArray **MP_**(sparseGridLevelSeq)
- static UShortArray **MP_**(tensorGridOrder)
- static UShortArray **MP_**(varPartitions)
- static String **MP_**(betaSolverName)
- static String **MP_**(dataDistFile)
- static String **MP_**(displayFormat)
- static String **MP_**(exportApproxPtsFile)
- static String **MP_**(exportCorrModelFile)
- static String **MP_**(exportCorrVarFile)
- static String **MP_**(exportDiscrepFile)
- static String **MP_**(exportExpansionFile)
- static String **MP_**(exportMCMCPtsFile)
- static String **MP_**(historyFile)
- static String **MP_**(hybridGlobalMethodName)
- static String **MP_**(hybridGlobalMethodPointer)
- static String **MP_**(hybridGlobalModelPointer)
- static String **MP_**(hybridLocalMethodName)
- static String **MP_**(hybridLocalMethodPointer)
- static String **MP_**(hybridLocalModelPointer)
- static String **MP_**(idMethod)
- static String **MP_**(importApproxPtsFile)
- static String **MP_**(importBuildPtsFile)
- static String **MP_**(importCandPtsFile)
12.1. DAKOTA NAMESPACE REFERENCE

- static String MP_ (importExpansionFile)
- static String MP_ (importPredConfigs)
- static String MP_ (logFile)
- static String MP_ (lowFidModelPointer)
- static String MP_ (modelPointer)
- static String MP_ (posteriorDensityExportFilename)
- static String MP_ (posteriorSamplesExportFilename)
- static String MP_ (posteriorSamplesImportFilename)
- static String MP_ (proposalCovFile)
- static String MP_ (pstudyFilename)
- static String MP_ (quesoOptionsFilename)
- static String MP_ (subMethodName)
- static String MP_ (subMethodPointer)
- static String MP_ (subModelPointer)
- static StringArray MP_ (hybridMethodNames)
- static StringArray MP_ (hybridMethodPointers)
- static StringArray MP_ (hybridModelPointers)
- static StringArray MP_ (miscOptions)
- static bool MP_ (adaptExpDesign)
- static bool MP_ (adaptPosteriorRefine)
- static bool MP_ (backfillFlag)
- static bool MP_ (calModelDiscrepancy)
- static bool MP_ (constantPenalty)
- static bool MP_ (crossValidation)
- static bool MP_ (crossValidNoiseOnly)
- static bool MP_ (dOptimal)
- static bool MP_ (evaluatePosteriorDensity)
- static bool MP_ (expansionFlag)
- static bool MP_ (exportSampleSeqFlag)
- static bool MP_ (fixedSeedFlag)
- static bool MP_ (fixedSequenceFlag)
- static bool MP_ (generatePosteriorSamples)
- static bool MP_ (gpmsaNormalize)
- static bool MP_ (importApproxActive)
- static bool MP_ (importBuildActive)
- static bool MP_ (latinizeFlag)
- static bool MP_ (logitTransform)
- static bool MP_ (mainEffectsFlag)
- static bool MP_ (methodScaling)
- static bool MP_ (methodUseDerivsFlag)
- static bool MP_ (mutualInfoKSG2)
- static bool MP_ (mutationAdaptive)
- static bool MP_ (normalizedCoeffs)
- static bool MP_ (pcaFlag)
- static bool MP_ (posteriorStatsKL)
- static bool MP_ (posteriorStatsMutual)
• static bool MP_ (printPopFlag)
• static bool MP_ (pstudyFileActive)
• static bool MP_ (randomizeOrderFlag)
• static bool MP_ (regressDiag)
• static bool MP_ (showAllEval)
• static bool MP_ (showMiscOptions)
• static bool MP_ (speculativeFlag)
• static bool MP_ (standardizedSpace)
• static bool MP_ (tensorGridFlag)
• static bool MP_ (surrBasedGlobalReplacePts)
• static bool MP_ (surrBasedLocalLayerBypass)
• static bool MP_ (vbdFlag)
• static bool MP_ (volQualityFlag)
• static bool MP_ (wilksFlag)
• static short MP_ (expansionType)
• static short MP_ (nestingOverride)
• static short MP_ (refinementType)
• static short MP_ (wilksSidedInterval)
• static int MP_ (buildSamples)
• static int MP_ (burnInSamples)
• static int MP_ (chainSamples)
• static int MP_ (concurrentRandomJobs)
• static int MP_ (contractAfterFail)
• static int MP_ (covarianceType)
• static int MP_ (crossoverChainPairs)
• static int MP_ (emulatorOrder)
• static int MP_ (expandAfterSuccess)
• static int MP_ (iteratorServers)
• static int MP_ (jumpStep)
• static int MP_ (maxFunctionEvaluations)
• static int MP_ (maxHifiEvals)
• static int MP_ (maxIterations)
• static int MP_ (maxRefineIterations)
• static int MP_ (maxSolverIterations)
• static int MP_ (mutationRange)
• static int MP_ (neighborOrder)
• static int MP_ (newSolnsGenerated)
• static int MP_ (numChains)
• static int MP_ (numCR)
• static int MP_ (numSamples)
• static int MP_ (numSteps)
• static int MP_ (numSymbols)
• static int MP_ (numTrials)
• static int MP_ (populationSize)
• static int MP_ (procsPerIterator)
• static int MP_ (proposalCovUpdates)
• static int MP_ (numPushforwardSamples)
• static int MP_ (randomSeed)
• static int MP_ (samplesOnEmulator)
• static int MP_ (searchSchemeSize)
• static int MP_ (subSamplingPeriod)
• static int MP_ (totalPatternSize)
• static int MP_ (verifyLevel)
• static size_t MP_ (collocationPoints)
• static size_t MP_ (expansionSamples)
• static size_t MP_ (numCandidateDesigns)
• static size_t MP_ (numCandidates)
• static size_t MP_ (numDesigns)
• static size_t MP_ (numFinalSolutions)
• static size_t MP_ (numGenerations)
• static size_t MP_ (numOffspring)
• static size_t MP_ (numParents)
• static size_t MP_ (numPredConfigs)
• static Method_mp_type MP2s (covarianceControl, DIAGONAL_COVARIANCE)
• static Method_mp_type MP2s (covarianceControl, FULL_COVARIANCE)
• static Method_mp_type MP2s (distributionType, COMPLEMENTARY)
• static Method_mp_type MP2s (distributionType, CUMULATIVE)
• static Method_mp_type MP2s (emulatorType, GP_EMULATOR)
• static Method_mp_type MP2s (emulatorType, KRIGING_EMULATOR)
• static Method_mp_type MP2s (emulatorType, MF_PCE_EMULATOR)
• static Method_mp_type MP2s (emulatorType, MF_SC_EMULATOR)
• static Method_mp_type MP2s (emulatorType, ML_PCE_EMULATOR)
• static Method_mp_type MP2s (emulatorType, PCE_EMULATOR)
• static Method_mp_type MP2s (emulatorType, SC_EMULATOR)
• static Method_mp_type MP2s (emulatorType, VPS_EMULATOR)
• static Method_mp_type MP2p (expansionBasisType, ADAPTED_BASIS_EXPANDING_FRONT)
• static Method_mp_type MP2p (expansionBasisType, ADAPTED_BASIS_GENERALIZED)
• static Method_mp_type MP2p (expansionBasisType, HIERARCHICAL_INTERPOLANT)
• static Method_mp_type MP2p (expansionBasisType, NODAL_INTERPOLANT)
• static Method_mp_type MP2p (expansionBasisType, TOTAL_ORDER_BASIS)
• static Method_mp_type MP2p (expansionType, ASKEY_U)
• static Method_mp_type MP2s (expansionType, STD_NORMAL_U)
• static Method_mp_type MP2s (finalMomentsType, CENTRAL_MOMENTS)
• static Method_mp_type MP2s (finalMomentsType, NO_MOMENTS)
• static Method_mp_type MP2s (finalMomentsType, STANDARD_MOMENTS)
• static Method_mp_type MP2p (growthOverride, RESTRICTED)
• static Method_mp_type MP2p (growthOverride, UNRESTRICTED)
• static Method_mp_type MP2s (iteratorScheduling, MASTER_SCHEDULING)
• static Method_mp_type MP2s (iteratorScheduling, PEER_SCHEDULING)
• static Method_mp_type MP2s (lsRegressionType, EQ_CON_LS)
• static Method_mp_type MP2s (lsRegressionType, SVD_LS)
• static Method\_mp\_type MP2o (meritFn, ArgaeZTapia)
• static Method\_mp\_type MP2o (meritFn, NormFmu)
• static Method\_mp\_type MP2o (meritFn, VanShanno)
• static Method\_mp\_type MP2s (methodOutput, DEBUG\_OUTPUT)
• static Method\_mp\_type MP2s (methodOutput, NORMAL\_OUTPUT)
• static Method\_mp\_type MP2s (methodOutput, QUIET\_OUTPUT)
• static Method\_mp\_type MP2s (methodOutput, SILENT\_OUTPUT)
• static Method\_mp\_type MP2s (methodOutput, VERBOSE\_OUTPUT)
• static Method\_mp\_type MP2s (multilevDiscrepEmulation, DISTINCT\_EMULATION)
• static Method\_mp\_type MP2s (multilevDiscrepEmulation, RECURSIVE\_EMULATION)
• static Method\_mp\_type MP2p (nestingOverride, NESTED)
• static Method\_mp\_type MP2p (nestingOverride, NON\_NESTED)
• static Method\_mp\_type MP2p (refinementControl, DIMENSION\_ADAPTIVE\_CONTROL\_GENERALIZED)
• static Method\_mp\_type MP2p (refinementControl, DIMENSION\_ADAPTIVE\_CONTROL\_DECAY)
• static Method\_mp\_type MP2p (refinementControl, DIMENSION\_ADAPTIVE\_CONTROL\_SOBOL)
• static Method\_mp\_type MP2p (refinementControl, LOCAL\_ADAPTIVE\_CONTROL)
• static Method\_mp\_type MP2p (refinementControl, UNIFORM\_CONTROL)
• static Method\_mp\_type MP2p (regressionType, P\_REFINEMENT)
• static Method\_mp\_type MP2p (regressionType, H\_REFINEMENT)
• static Method\_mp\_type MP2p (regressionType, BASIS\_PURSUIT)
• static Method\_mp\_type MP2p (regressionType, BASIS\_PURSUIT\_DENOISING)
• static Method\_mp\_type MP2p (regressionType, DEFAULT\_LEAST\_SQ\_REGRESSION)
• static Method\_mp\_type MP2p (regressionType, LASSO\_REGRESSION)
• static Method\_mp\_type MP2p (regressionType, LEAST\_ANGLE\_REGRESSION)
• static Method\_mp\_type MP2p (regressionType, ORTHOG\_LEAST\_INTERPOLATION)
• static Method\_mp\_type MP2p (regressionType, ORTHOG\_MATCH\_PURSUIT)
• static Method\_mp\_type MP2s (responseLevelTarget, GEN\_RELIABILITIES)
• static Method\_mp\_type MP2s (responseLevelTarget, PROBABILITIES)
• static Method\_mp\_type MP2s (responseLevelTarget, RELIABILITIES)
• static Method\_mp\_type MP2s (responseLevelTargetReduce, SYSTEM\_PARALLEL)
• static Method\_mp\_type MP2s (responseLevelTargetReduce, SYSTEM\_SERIES)
• static Method\_mp\_type MP2s (surrBasedLocalAcceptLogic, FILTER)
• static Method\_mp\_type MP2s (surrBasedLocalAcceptLogic, TR\_RATIO)
• static Method\_mp\_type MP2s (surrBasedLocalConstrRelax, HOMOTOPIE)
• static Method\_mp\_type MP2s (surrBasedLocalMeritFn, ADAPTIVE\_PENALTY\_MERIT)
• static Method\_mp\_type MP2s (surrBasedLocalMeritFn, AUGMENTED\_LAGRANGIAN\_MERIT)
• static Method\_mp\_type MP2s (surrBasedLocalMeritFn, LAGRANGIAN\_MERIT)
• static Method\_mp\_type MP2s (surrBasedLocalMeritFn, PENALTY\_MERIT)
• static Method\_mp\_type MP2s (surrBasedLocalSubProbCon, LINEARIZED\_CONSTRAINTS)
• static Method\_mp\_type MP2s (surrBasedLocalSubProbCon, NO\_CONSTRAINTS)
• static Method\_mp\_type MP2s (surrBasedLocalSubProbCon, ORIGINAL\_CONSTRAINTS)
• static Method\_mp\_type MP2s (surrBasedLocalSubProbObj, AUGMENTED\_LAGRANGIAN\_OBJECTIVE)
• static Method\_mp\_type MP2s (surrBasedLocalSubProbObj, LAGRANGIAN\_OBJECTIVE)
• static Method\_mp\_type MP2s (surrBasedLocalSubProbObj, ORIGINAL\_PRIMARY)
- static Method 
  - static Method_mp_type MP2s (surrBasedLocalSubProbObj, SINGLE_OBJECTIVE)
  - static Method_mp_type MP2s (wilksSidedInterval, ONE_SIDED_LOWER)
  - static Method_mp_type MP2s (wilksSidedInterval, ONE_SIDED_UPPER)
  - static Method_mp_type MP2s (wilksSidedInterval, TWO_SIDED)
  - static Method_mp_utype MP2s (calibrateErrorMode, CALIBRATE_ONE)
  - static Method_mp_utype MP2s (calibrateErrorMode, CALIBRATE_PER_EXPER)
  - static Method_mp_utype MP2s (calibrateErrorMode, CALIBRATE_PER_RESP)
  - static Method_mp_utype MP2s (calibrateErrorMode, CALIBRATE_BOTH)
  - static Method_mp_utype MP2s (exportApproxFormat, TABULARNONE)
  - static Method_mp_utype MP2s (exportApproxFormat, TABULARHEADER)
  - static Method_mp_utype MP2s (exportApproxFormat, TABULAR_EVAL_ID)
  - static Method_mp_utype MP2s (exportApproxFormat, TABULAR_IFACE_ID)
  - static Method_mp_utype MP2s (exportApproxFormat, TABULARANNOTATED)
  - static Method_mp_utype MP2s (exportCorrModelFormat, TABULARNONE)
  - static Method_mp_utype MP2s (exportCorrModelFormat, TABULARHEADER)
  - static Method_mp_utype MP2s (exportCorrModelFormat, TABULAR_EVAL_ID)
  - static Method_mp_utype MP2s (exportCorrModelFormat, TABULAR_IFACE_ID)
  - static Method_mp_utype MP2s (exportCorrModelFormat, TABULARANNOTATED)
  - static Method_mp_utype MP2s (exportCorrVarFormat, TABULARNONE)
  - static Method_mp_utype MP2s (exportCorrVarFormat, TABULARHEADER)
  - static Method_mp_utype MP2s (exportCorrVarFormat, TABULAR_EVAL_ID)
  - static Method_mp_utype MP2s (exportCorrVarFormat, TABULAR_IFACE_ID)
  - static Method_mp_utype MP2s (exportCorrVarFormat, TABULARANNOTATED)
  - static Method_mp_utype MP2s (exportDiscrepFormat, TABULARNONE)
  - static Method_mp_utype MP2s (exportDiscrepFormat, TABULARHEADER)
  - static Method_mp_utype MP2s (exportDiscrepFormat, TABULAR_EVAL_ID)
  - static Method_mp_utype MP2s (exportDiscrepFormat, TABULAR_IFACE_ID)
  - static Method_mp_utype MP2s (exportDiscrepFormat, TABULARANNOTATED)
  - static Method_mp_utype MP2s (exportSamplesFormat, TABULARNONE)
  - static Method_mp_utype MP2s (exportSamplesFormat, TABULARHEADER)
  - static Method_mp_utype MP2s (exportSamplesFormat, TABULAR_EVAL_ID)
  - static Method_mp_utype MP2s (exportSamplesFormat, TABULAR_IFACE_ID)
  - static Method_mp_utype MP2s (exportSamplesFormat, TABULARANNOTATED)
  - static Method_mp_utype MP2s (importApproxFormat, TABULARNONE)
  - static Method_mp_utype MP2s (importApproxFormat, TABULARHEADER)
  - static Method_mp_utype MP2s (importApproxFormat, TABULAR_EVAL_ID)
  - static Method_mp_utype MP2s (importApproxFormat, TABULAR_IFACE_ID)
  - static Method_mp_utype MP2s (importCandFormat, TABULARNONE)
  - static Method_mp_utype MP2s (importCandFormat, TABULARHEADER)
• static Method_mp_utype MP2s (importCandFormat, TABULAR_IFACE_ID)
• static Method_mp_utype MP2s (importCandFormat, TABULAR_ANNOTATED)
• static Method_mp_utype MP2s (importPredConfigFormat, TABULAR_NONE)
• static Method_mp_utype MP2s (importPredConfigFormat, TABULAR_HEADER)
• static Method_mp_utype MP2s (importPredConfigFormat, TABULAR_EVAL_ID)
• static Method_mp_utype MP2s (importPredConfigFormat, TABULAR_IFACE_ID)
• static Method_mp_utype MP2s (importPredConfigFormat, TABULAR_ANNOTATED)
• static Method_mp_utype MP2s (integrationRefine, AIS)
• static Method_mp_utype MP2s (integrationRefine, IS)
• static Method_mp_utype MP2s (integrationRefine, MMAIS)
• static Method_mp_utype MP2s (methodName, ASYNCH_PATTERN_SEARCH)
• static Method_mp_utype MP2s (methodName, BRANCH_AND_BOUND)
• static Method_mp_utype MP2s (methodName, COLINY_BETA)
• static Method_mp_utype MP2s (methodName, COLINY_COBYLA)
• static Method_mp_utype MP2s (methodName, COLINY_DIRECT)
• static Method_mp_utype MP2s (methodName, COLINY_EA)
• static Method_mp_utype MP2s (methodName, COLINY_PATTERN_SEARCH)
• static Method_mp_utype MP2s (methodName, COLINY_SOLIS_WETS)
• static Method_mp_utype MP2s (methodName, CONMIN_FRCG)
• static Method_mp_utype MP2s (methodName, CONMIN_MFD)
• static Method_mp_utype MP2s (methodName, DACE)
• static Method_mp_utype MP2s (methodName, DATA_FIT_SURROGATE_BASED_LOCAL)
• static Method_mp_utype MP2s (methodName, DOT_BFGS)
• static Method_mp_utype MP2s (methodName, DOT_FRCG)
• static Method_mp_utype MP2s (methodName, DOT_MMFD)
• static Method_mp_utype MP2s (methodName, DOT_SLP)
• static Method_mp_utype MP2s (methodName, DOT_SQP)
• static Method_mp_utype MP2s (methodName, EFFICIENT_GLOBAL)
• static Method_mp_utype MP2s (methodName, FSU_CVT)
• static Method_mp_utype MP2s (methodName, FSU_HALTON)
• static Method_mp_utype MP2s (methodName, FSU_HAMMERSLEY)
• static Method_mp_utype MP2s (methodName, HIERARCH_SURROGATE_BASED_LOCAL)
• static Method_mp_utype MP2s (methodName, HYBRID)
• static Method_mp_utype MP2s (methodName, MESH_ADAPTIVE_SEARCH)
• static Method_mp_utype MP2s (methodName, MOGA)
• static Method_mp_utype MP2s (methodName, MULTI_START)
• static Method_mp_utype MP2s (methodName, NCSU_DIRECT)
• static Method_mp_utype MP2s (methodName, NL2SOL)
• static Method_mp_utype MP2s (methodName, NLPQL_SQP)
• static Method_mp_utype MP2s (methodName, NLLSOL_SQP)
• static Method_mp_utype MP2s (methodName, MIT_NOWPAC)
• static Method_mp_utype MP2s (methodName, MIT_SNOWPAC)
• static Method_mp_utype MP2s (methodName, ADAPTIVE_SAMPLING)
• static Method_mp_utype MP2s (methodName, BAYES_CALIBRATION)
• static Method_mp_utype MP2s (methodName, GENIE_DIRECT)
• static Method_mp_utype MP2s (methodName, GENIE_OPT_DARTS)
12.1. DAKOTA NAMESPACE REFERENCE

- static Method mp_utype MP2s (methodName, GPAIS)
- static Method mp_utype MP2s (methodName, GLOBAL_EVIDENCE)
- static Method mp_utype MP2s (methodName, GLOBAL_INTERVAL_EST)
- static Method mp_utype MP2s (methodName, GLOBAL_RELIABILITY)
- static Method mp_utype MP2s (methodName, IMPORTANCE_SAMPLING)
- static Method mp_utype MP2s (methodName, LOCAL_EVIDENCE)
- static Method mp_utype MP2s (methodName, LOCAL_INTERVAL_EST)
- static Method mp_utype MP2s (methodName, LOCAL_RELIABILITY)
- static Method mp_utype MP2s (methodName, MULTIFIDELITY_POLYNOMIAL_CHAOS)
- static Method mp_utype MP2s (methodName, MULTIFIDELITY_STOCH_COLLOCATION)
- static Method mp_utype MP2s (methodName, MULTILEVEL_POLYNOMIAL_CHAOS)
- static Method mp_utype MP2s (methodName, MULTILEVEL_SAMPLING)
- static Method mp_utype MP2s (methodName, POF_DARTS)
- static Method mp_utype MP2s (methodName, RKD_DARTS)
- static Method mp_utype MP2s (methodName, POLYNOMIAL_CHAOS)
- static Method mp_utype MP2s (methodName, RANDOM_SAMPLING)
- static Method mp_utype MP2s (methodName, STOCH_COLLOCATION)
- static Method mp_utype MP2s (methodName, NONLINEAR_CG)
- static Method mp_utype MP2s (methodName, NPSOL_SQP)
- static Method mp_utype MP2s (methodName, OPTPP_CG)
- static Method mp_utype MP2s (methodName, OPTPP_FD_NEWTON)
- static Method mp_utype MP2s (methodName, OPTPP_G_NEWTON)
- static Method mp_utype MP2s (methodName, OPTPP_NEWTON)
- static Method mp_utype MP2s (methodName, OPTPP_PDS)
- static Method mp_utype MP2s (methodName, OPTPP_Q_NEWTON)
- static Method mp_utype MP2s (methodName, PARETO_SET)
- static Method mp_utype MP2s (methodName, PSUADE_MOAT)
- static Method mp_utype MP2s (methodName, RICHARDSON_EXTRAP)
- static Method mp_utype MP2s (methodName, SOGA)
- static Method mp_utype MP2s (methodName, SURROGATE_BASED_GLOBAL)
- static Method mp_utype MP2s (methodName, SURROGATE_BASED_LOCAL)
- static Method mp_utype MP2s (methodName, VECTOR_PARAMETER_STUDY)
- static Method mp_utype MP2s (methodName, LIST_PARAMETER_STUDY)
- static Method mp_utype MP2s (methodName, CENTERED_PARAMETER_STUDY)
- static Method mp_utype MP2s (methodName, MULTIDIM_PARAMETER_STUDY)
- static Method mp_utype MP2s (preSolveMethod, SUBMETHOD_NIP)
- static Method mp_utype MP2s (preSolveMethod, SUBMETHOD_NONE)
- static Method mp_utype MP2s (preSolveMethod, SUBMETHOD_SQP)
- static Method mp_utype MP2s (pstudyFileFormat, TABULAR_NONE)
- static Method mp_utype MP2s (pstudyFileFormat, TABULAR_HEADER)
- static Method mp_utype MP2s (pstudyFileFormat, TABULAR_IFACE_ID)
- static Method mp_utype MP2s (pstudyFileFormat, TABULAR_ANNOTATED)
- static Method mp_utype MP2s (reliabilitySearchType, AMV_PLUS_U)
- static Method mp_utype MP2s (reliabilitySearchType, AMV_PLUS_X)
- static Method mp_utype MP2s (reliabilitySearchType, AMV_U)
• static Method\_mp\_utype \texttt{MP2s} (reliabilitySearchType, AMV\_X)
• static Method\_mp\_utype \texttt{MP2s} (reliabilitySearchType, EGRA\_U)
• static Method\_mp\_utype \texttt{MP2s} (reliabilitySearchType, EGRA\_X)
• static Method\_mp\_utype \texttt{MP2s} (reliabilitySearchType, NO\_APPROX)
• static Method\_mp\_utype \texttt{MP2s} (reliabilitySearchType, TANA\_U)
• static Method\_mp\_utype \texttt{MP2s} (reliabilitySearchType, TANA\_X)
• static Method\_mp\_utype \texttt{MP2s} (sampleType, SUBMETHOD\_LHS)
• static Method\_mp\_utype \texttt{MP2s} (sampleType, SUBMETHOD\_RANDOM)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_COLLABORATIVE)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_EMBEDDED)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_SEQUENTIAL)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_DREAM)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_WASABI)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_GPMSA)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_QUESO)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_NIP)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_SQP)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_EA)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_EGO)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_SBO)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_LHS)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_RANDOM)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_OA\_LHS)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_OAS)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_BOX\_BEHNKEN)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_CENTRAL\_COMPOSITE)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_GRID)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_CONVERGE\_ORDER)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_CONVERGE\_QOI)
• static Method\_mp\_utype \texttt{MP2s} (subMethod, SUBMETHOD\_ESTIMATE\_ORDER)
• static IntSet \texttt{MP2} (surrogateFnIndices)
• static Model\_mp\_lit \texttt{MP2} (approxPointReuse, all)
• static Model\_mp\_lit \texttt{MP2} (approxPointReuse, none)
• static Model\_mp\_lit \texttt{MP2} (approxPointReuse, region)
• static Model\_mp\_lit \texttt{MP2} (marsInterpolation, linear)
• static Model\_mp\_lit \texttt{MP2} (marsInterpolation, cubic)
• static Model\_mp\_lit \texttt{MP2} (modelType, active\_subspace)
• static Model\_mp\_lit \texttt{MP2} (modelType, adapted\_basis)
• static Model\_mp\_lit \texttt{MP2} (modelType, nested)
• static Model\_mp\_lit \texttt{MP2} (modelType, random\_field)
• static Model\_mp\_lit \texttt{MP2} (modelType, simulation)
• static Model\_mp\_lit \texttt{MP2} (modelType, surrogate)
• static Model\_mp\_lit \texttt{MP2} (surrogateType, hierarchical)
• static Model\_mp\_lit \texttt{MP2} (surrogateType, global\_gaussian)
• static Model\_mp\_lit \texttt{MP2} (surrogateType, global\_kriging)
• static Model\_mp\_lit \texttt{MP2} (surrogateType, global\_mars)
• static Model_mp_lit MP2 (surrogateType, global_moving_least_squares)
• static Model_mp_lit MP2 (surrogateType, global_neural_network)
• static Model_mp_lit MP2 (surrogateType, global_polynomial)
• static Model_mp_lit MP2 (surrogateType, global_radial_basis)
• static Model_mp_lit MP2 (surrogateType, global_voronoi_surrogate)
• static Model_mp_lit MP2 (surrogateType, local_taylor)
• static Model_mp_lit MP2 (trendOrder, constant)
• static Model_mp_lit MP2 (trendOrder, linear)
• static Model_mp_lit MP2 (trendOrder, reduced_quadratic)
• static Model_mp_lit MP2 (trendOrder, quadratic)
• static Model_mp_ord MP2s (polynomialOrder, 1)
• static Model_mp_ord MP2s (polynomialOrder, 2)
• static Model_mp_ord MP2s (polynomialOrder, 3)
• static Model_mp_type MP2s (approxCorrectionType, ADDITIVE_CORRECTION)
• static Model_mp_type MP2s (approxCorrectionType, COMBINED_CORRECTION)
• static Model_mp_type MP2s (approxCorrectionType, MULTIPLICATIVE_CORRECTION)
• static Model_mp_type MP2s (pointsManagement, MINIMUM_POINTS)
• static Model_mp_type MP2s (pointsManagement, RECOMMENDED_POINTS)
• static Model_mp_type MP2s (subMethodScheduling, MASTER_SCHEDULING)
• static Model_mp_type MP2s (subMethodScheduling, PEER_SCHEDULING)
• static Model_mp_utype MP2s (analyticCovIdForm, EXP_L2)
• static Model_mp_utype MP2s (analyticCovIdForm, EXP_L1)
• static Model_mp_utype MP2s (importChallengeFormat, TABULAR_NONE)
• static Model_mp_utype MP2s (importChallengeFormat, TABULAR_HEADER)
• static Model_mp_utype MP2s (importChallengeFormat, TABULAR_EVAL_ID)
• static Model_mp_utype MP2s (importChallengeFormat, TABULAR_IFACE_ID)
• static Model_mp_utype MP2s (importChallengeFormat, TABULAR_ANNOTATED)
• static Model_mp_utype MP2s (modelExportFormat, TEXT_ARCHIVE)
• static Model_mp_utype MP2s (modelExportFormat, BINARY_ARCHIVE)
• static Model_mp_utype MP2s (modelExportFormat, ALGEBRAIC_FILE)
• static Model_mp_utype MP2s (modelExportFormat, ALGEBRAIC_CONSOLE)
• static Model_mp_utype MP2s (randomFieldIdForm, RF_KARHUNEN_LOEVE)
• static Model_mp_utype MP2s (randomFieldIdForm, RF_PCA_GP)
• static Model_mp_utype MP2s (subspaceNormalization, SUBSPACE_NORM_MEAN_VALUE)
• static Model_mp_utype MP2s (subspaceNormalization, SUBSPACE_NORM_MEAN_GRAD)
• static Model_mp_utype MP2s (subspaceNormalization, SUBSPACE_NORM_LOCAL_GRAD)
• static Model_mp_utype MP2s (subspaceSampleType, SUBMETHOD_LHS)
• static Model_mp_utype MP2s (subspaceSampleType, SUBMETHOD_RANDOM)
• static Model_mp_utype MP2s (subspaceIdCVMethod, MINIMUM_METRIC)
• static Model_mp_utype MP2s (subspaceIdCVMethod, RELATIVE_TOLERANCE)
• static Model_mp_utype MP2s (subspaceIdCVMethod, DECREASE_TOLERANCE)
• static Real MP_ (adaptedBasisCollocRatio)
• static Real MP_ (annRange)
• static Real MP_ (discontGradThresh)
• static Real MP_ (discontJumpThresh)
• static Real MP_ (krigingNugget)
• static Real MP_ (percentFold)
• static Real MP_ (truncationTolerance)
• static Real MP_ (relTolerance)
• static Real MP_ (decreaseTolerance)
• static RealVector MP_ (krigingCorrelations)
• static RealVector MP_ (primaryRespCoeffs)
• static RealVector MP_ (secondaryRespCoeffs)
• static RealVector MP_ (solutionLevelCost)
• static String MP_ (actualModelPointer)
• static String MP_ (decompCellType)
• static String MP_ (idModel)
• static String MP_ (importChallengePtsFile)
• static String MP_ (interfacePointer)
• static String MP_ (krigingOptMethod)
• static String MP_ (modelExportPrefix)
• static String MP_ (optionalInterfRespPointer)
• static String MP_ (propagationModelPointer)
• static String MP_ (refineCVMetric)
• static String MP_ (responsesPointer)
• static String MP_ (rfDataFileName)
• static String MP_ (solutionLevelControl)
• static String MP_ (variablesPointer)
• static StringArray MP_ (diagMetrics)
• static StringArray MP_ (orderedModelPointers)
• static StringArray MP_ (primaryVarMaps)
• static StringArray MP_ (secondaryVarMaps)
• static bool MP_ (autoRefine)
• static bool MP_ (crossValidateFlag)
• static bool MP_ (decompDiscontDetect)
• static bool MP_ (exportSurrogate)
• static bool MP_ (hierarchicalTags)
• static bool MP_ (importChallengeActive)
• static bool MP_ (modelUseDerivsFlag)
• static bool MP_ (domainDecomp)
• static bool MP_ (pointSelection)
• static bool MP_ (pressFlag)
• static bool MP_ (subspaceIdBingLi)
• static bool MP_ (subspaceIdConstantine)
• static bool MP_ (subspaceIdEnergy)
• static bool MP_ (subspaceBuildSurrogate)
• static bool MP_ (subspaceIdCV)
• static bool MP_ (subspaceCVIncremental)
• static unsigned short MP_ (adaptedBasisSparseGridLev)
• static unsigned short MP_ (adaptedBasisExpOrder)
• static short MP_ (annNodes)
12.1. DAKOTA NAMESPACE REFERENCE

- static short \texttt{MP\_}(annRandomWeight)
- static short \texttt{MP\_}(krigingFindNugget)
- static short \texttt{MP\_}(krigingMaxTrials)
- static short \texttt{MP\_}(marsMaxBases)
- static short \texttt{MP\_}(mlsWeightFunction)
- static short \texttt{MP\_}(polynomialOrder)
- static short \texttt{MP\_}(rbfBases)
- static short \texttt{MP\_}(rbfMaxPts)
- static short \texttt{MP\_}(rbfMaxSubsets)
- static short \texttt{MP\_}(rbfMinPartition)
- static int \texttt{MP\_}(decompSupportLayers)
- static int \texttt{MP\_}(initialSamples)
- static int \texttt{MP\_}(maxFunctionEvals)
- static int \texttt{MP\_}(numFolds)
- static int \texttt{MP\_}(numReplicates)
- static int \texttt{MP\_}(pointsTotal)
- static int \texttt{MP\_}(refineCVFolds)
- static int \texttt{MP\_}(softConvergenceLimit)
- static int \texttt{MP\_}(subMethodProcs)
- static int \texttt{MP\_}(subMethodServers)
- static int \texttt{MP\_}(subspaceDimension)
- static int \texttt{MP\_}(subspaceCVMaxRank)
- static \texttt{IntSet}\texttt{MP\_}(idAnalyticGrads)
- static \texttt{IntSet}\texttt{MP\_}(idAnalyticHessians)
- static \texttt{IntSet}\texttt{MP\_}(idNumericalGrads)
- static \texttt{IntSet}\texttt{MP\_}(idNumericalHessians)
- static \texttt{IntSet}\texttt{MP\_}(idQuasiHessians)
- static \texttt{IntVector}\texttt{MP\_}(fieldLengths)
- static \texttt{IntVector}\texttt{MP\_}(numCoordsPerField)
- static \texttt{RealVector}\texttt{MP\_}(expConfigVars)
- static \texttt{RealVector}\texttt{MP\_}(expObservations)
- static \texttt{RealVector}\texttt{MP\_}(primaryRespFnWeights)
- static \texttt{RealVector}\texttt{MP\_}(nonlinearEqTargets)
- static \texttt{RealVector}\texttt{MP\_}(nonlinearIneqLowerBnds)
- static \texttt{RealVector}\texttt{MP\_}(nonlinearIneqUpperBnds)
- static \texttt{RealVector}\texttt{MP\_}(simVariance)
- static \texttt{RealVector}\texttt{MP\_}(fdGradStepSize)
- static \texttt{RealVector}\texttt{MP\_}(fdHessStepSize)
- static \texttt{RealVector}\texttt{MP\_}(primaryRespFnScales)
- static \texttt{RealVector}\texttt{MP\_}(nonlinearEqScales)
- static \texttt{RealVector}\texttt{MP\_}(nonlinearIneqScales)
- static \texttt{Resp\_mp\_lit}\texttt{MP2}(gradientType, analytic)
- static \texttt{Resp\_mp\_lit}\texttt{MP2}(gradientType, mixed)
- static \texttt{Resp\_mp\_lit}\texttt{MP2}(gradientType, none)
- static \texttt{Resp\_mp\_lit}\texttt{MP2}(gradientType, numerical)
- static \texttt{Resp\_mp\_lit}\texttt{MP2}(hessianType, analytic)
• static Resp_mp_lit MP2 (hessianType, mixed)
• static Resp_mp_lit MP2 (hessianType, none)
• static Resp_mp_lit MP2 (hessianType, numerical)
• static Resp_mp_lit MP2 (hessianType, quasi)
• static Resp_mp_lit MP2 (intervalType, central)
• static Resp_mp_lit MP2 (intervalType, forward)
• static Resp_mp_lit MP2 (methodSource, dakota)
• static Resp_mp_lit MP2 (methodSource, vendor)
• static Resp_mp_lit MP2 (fdGradStepType, absolute)
• static Resp_mp_lit MP2 (fdGradStepType, bounds)
• static Resp_mp_lit MP2 (fdGradStepType, relative)
• static Resp_mp_lit MP2 (fdHessStepType, absolute)
• static Resp_mp_lit MP2 (fdHessStepType, bounds)
• static Resp_mp_lit MP2 (fdHessStepType, relative)
• static Resp_mp_lit MP2 (quasiHessianType, bfgs)
• static Resp_mp_lit MP2 (quasiHessianType, damped_bfgs)
• static Resp_mp_lit MP2 (quasiHessianType, sr1)
• static String MP_ (scalarDataFileName)
• static String MP_ (idResponses)
• static StringArray MP_ (nonlinearEqScaleTypes)
• static StringArray MP_ (nonlinearIneqScaleTypes)
• static StringArray MP_ (primaryRespFnScaleTypes)
• static StringArray MP_ (primaryRespFnSense)
• static StringArray MP_ (responseLabels)
• static String MP_ (varianceType)
• static bool MP_ (calibrationDataFlag)
• static bool MP_ (centralHess)
• static bool MP_ (interpolateFlag)
• static bool MP_ (ignoreBounds)
• static bool MP_ (readFieldCoords)
• static size_t MP_ (numExpConfigVars)
• static size_t MP_ (numExperiments)
• static size_t MP_ (numFieldLeastSqTerms)
• static size_t MP_ (numFieldObjectiveFunctions)
• static size_t MP_ (numFieldResponseFunctions)
• static size_t MP_ (numLeastSqTerms)
• static size_t MP_ (numNonlinearEqConstraints)
• static size_t MP_ (numNonlinearIneqConstraints)
• static size_t MP_ (numObjectiveFunctions)
• static size_t MP_ (numResponseFunctions)
• static size_t MP_ (numScalarLeastSqTerms)
• static size_t MP_ (numScalarObjectiveFunctions)
• static size_t MP_ (numScalarResponseFunctions)
• static Resp_mp_utype MP2s (scalarDataFormat, TABULAR_NONE)
• static Resp_mp_utype MP2s (scalarDataFormat, TABULAR_HEADER)
• static Resp_mp_utype MP2s (scalarDataFormat, TABULAR_EVAL_ID)
- static Resp_mp_utype MP2s (scalarDataFormat, TABULAR_EXPER_ANNOT)
- static Env_mp_utype MP2s (postRunInputFormat, TABULAR_NONE)
- static Env_mp_utype MP2s (postRunInputFormat, TABULAR_HEADER)
- static Env_mp_utype MP2s (postRunInputFormat, TABULAR_EVAL_ID)
- static Env_mp_utype MP2s (postRunInputFormat, TABULAR_IFACE_ID)
- static Env_mp_utype MP2s (postRunInputFormat, TABULAR_ANNOTATED)
- static Env_mp_utype MP2s (preRunOutputFormat, TABULAR_NONE)
- static Env_mp_utype MP2s (preRunOutputFormat, TABULAR_HEADER)
- static Env_mp_utype MP2s (preRunOutputFormat, TABULAR_EVAL_ID)
- static Env_mp_utype MP2s (preRunOutputFormat, TABULAR_IFACE_ID)
- static Env_mp_utype MP2s (preRunOutputFormat, TABULAR_ANNOTATED)
- static Env_mp_utype MP2s (tabularFormat, TABULAR_NONE)
- static Env_mp_utype MP2s (tabularFormat, TABULAR_HEADER)
- static Env_mp_utype MP2s (tabularFormat, TABULAR_EVAL_ID)
- static Env_mp_utype MP2s (tabularFormat, TABULAR_IFACE_ID)
- static Env_mp_utype MP2s (tabularFormat, TABULAR_ANNOTATED)
- static String MP_ (errorFile)
- static String MP_ (outputFile)
- static String MP_ (runInput)
- static String MP_ (runOutput)
- static String MP_ (tabularDataFile)
- static String MP_ (topMethodPointer)
- static String MP_ (writeRestart)
- static bool MP_ (checkFlag)
- static bool MP_ (graphicsFlag)
- static bool MP_ (postRunFlag)
- static bool MP_ (preRunFlag)
- static bool MP_ (resultsOutputFlag)
- static bool MP_ (runFlag)
- static bool MP_ (tabularDataFlag)
- static int MP_ (outputPrecision)
- static int MP_ (stopRestart)
- static size_t MP_ (numBetaUncVars)
- static size_t MP_ (numBinomialUncVars)
- static size_t MP_ (numContinuousDesVars)
- static size_t MP_ (numContinuousIntervalUncVars)
- static size_t MP_ (numContinuousStateVars)
- static size_t MP_ (numDiscreteDesRangeVars)
- static size_t MP_ (numDiscreteDesSetIntVars)
- static size_t MP_ (numDiscreteDesSetStrVars)
- static size_t MP_ (numDiscreteDesSetRealVars)
- static size_t MP_ (numDiscreteIntervalUncVars)
- static size_t MP_ (numDiscreteStateRangeVars)
- static size_t MP_ (numDiscreteStateSetIntVars)
- static size_t MP_ (numDiscreteStateSetStrVars)
- static size_t MP_ (numDiscreteStateSetRealVars)
- static size_t MP_ (numExponentialUncVars)
- static size_t MP_ (numFrechetUncVars)
- static size_t MP_ (numGammaUncVars)
- static size_t MP_ (numGeometricUncVars)
- static size_t MP_ (numGumbelUncVars)
- static size_t MP_ (numHistogramBinUncVars)
- static size_t MP_ (numHistogramPtIntUncVars)
- static size_t MP_ (numHistogramPtStrUncVars)
- static size_t MP_ (numHistogramPtRealUncVars)
- static size_t MP_ (numHyperGeomUncVars)
- static size_t MP_ (numLognormalUncVars)
- static size_t MP_ (numLoguniformUncVars)
- static size_t MP_ (numNegBinomialUncVars)
- static size_t MP_ (numNormalUncVars)
- static size_t MP_ (numPoissonUncVars)
- static size_t MP_ (numTriangularUncVars)
- static size_t MP_ (numUniformUncVars)
- static size_t MP_ (numWeibullUncVars)
- static IntVector VP_ (ddsi)
- static IntVector VP_ (DIlb)
- static IntVector MP_ (discreteDesignRangeLowerBnds)
- static IntVector MP_ (discreteDesignRangeUpperBnds)
- static IntVector MP_ (discreteDesignRangeVars)
- static IntVector MP_ (discreteDesignSetIntVars)
- static IntVector MP_ (discreteIntervalUncVars)
- static IntVector MP_ (discreteStateRangeLowerBnds)
- static IntVector MP_ (discreteStateRangeUpperBnds)
- static IntVector MP_ (discreteStateRangeVars)
- static IntVector MP_ (discreteStateSetIntVars)
- static IntVector MP_ (discreteUncSetIntVars)
- static IntVector VP_ (Dlub)
- static IntVector MP_ (histogramPointIntUncVars)
- static IntVector VP_ (hpia)
- static IntVector VP_ (dssi)
- static IntVector VP_ (ddsia)
- static IntVector VP_ (ddssa)
- static IntVector VP_ (ddsra)
12.1. DAKOTA NAMESPACE REFERENCE

- static IntVector VP (dusi)
- static IntArray VP (nddsi)
- static IntArray VP (nddss)
- static IntArray VP (nddsr)
- static IntArray VP (ndssi)
- static IntArray VP (ndss)
- static IntArray VP (ndssr)
- static IntArray VP (ndusi)
- static IntArray VP (nduss)
- static IntArray VP (ndusr)
- static IntArray VP (nhbp)
- static IntArray VP (nhibp)
- static IntArray VP (nhpsp)
- static IntArray VP (nhprp)
- static IntArray VP (nCI)
- static IntArray VP (nDI)
- static RealVector MP (betaUncLowerBnds)
- static RealVector MP (betaUncUpperBnds)
- static RealVector MP (betaUncVars)
- static RealVector MP (binomialUncProbPerTrial)
- static RealVector MP (continuousDesignLowerBnds)
- static RealVector MP (continuousDesignUpperBnds)
- static RealVector MP (continuousDesignVars)
- static RealVector MP (continuousDesignScales)
- static RealVector MP (continuousIntervalUncVars)
- static RealVector MP (continuousStateLowerBnds)
- static RealVector MP (continuousStateUpperBnds)
- static RealVector MP (continuousStateVars)
- static RealVector MP (discreteDesignSetRealVars)
- static RealVector MP (discreteStateSetRealVars)
- static RealVector MP (discreteUncSetRealVars)
- static RealVector MP (frechetUncBetans)
- static RealVector MP (frechetUncVars)
- static RealVector MP (geometricUncProbPerTrial)
- static RealVector MP (gumbelUncBetans)
- static RealVector MP (gumbelUncVars)
- static RealVector MP (histogramBinUncVars)
- static RealVector MP (histogramPointRealUncVars)
- static RealVector MP (linearEqConstraintCoeffs)
- static RealVector MP (linearEqScales)
- static RealVector MP (linearEqTargets)
- static RealVector MP (linearIneqConstraintCoeffs)
- static RealVector MP (linearIneqLowerBnds)
- static RealVector MP (linearIneqUpperBnds)
- static RealVector MP (linearIneqScales)
- static RealVector MP (negBinomialUncProbPerTrial)
• static RealVector MP_ (normalUncLowerBnds)
• static RealVector MP_ (normalUncMeans)
• static RealVector MP_ (normalUncUpperBnds)
• static RealVector MP_ (normalUncVars)
• static RealVector MP_ (triangularUncModes)
• static RealVector MP_ (triangularUncVars)
• static RealVector MP_ (uniformUncVars)
• static RealVector MP_ (weibullUncVars)
• static RealVector VP_ (ddsr)
• static RealVector VP_ (dssr)
• static RealVector VP_ (dusr)
• static RealVector VP_ (CIIlb)
• static RealVector VP_ (CIIlb)
• static RealVector VP_ (CIIp)
• static RealVector VP_ (DIIp)
• static RealVector VP_ (DSIIp)
• static RealVector VP_ (DSSIIp)
• static RealVector VP_ (DSIIp)
• static RealVector VP_ (hba)
• static RealVector VP_ (hbo)
• static RealVector VP_ (hbc)
• static RealVector VP_ (hpic)
• static RealVector VP_ (hpsc)
• static RealVector VP_ (hptra)
• static RealVector VP_ (hptra)
• static RealVector VP_ (ucm)
• static String MP_ (idVariables)
• static StringArray MP_ (continuousDesignLabels)
• static StringArray MP_ (continuousDesignScaleTypes)
• static StringArray MP_ (continuousStateLabels)
• static StringArray MP_ (discreteDesignRangeLabels)
• static StringArray MP_ (discreteDesignSetIntLabels)
• static StringArray MP_ (discreteDesignSetStrLabels)
• static StringArray MP_ (discreteDesignSetRealLabels)
• static StringArray MP_ (discreteStateRangeLabels)
• static StringArray MP_ (discreteStateSetIntLabels)
• static StringArray MP_ (discreteStateSetStrLabels)
• static StringArray MP_ (discreteStateSetRealLabels)
• static StringArray MP_ (discreteUncSetIntVars)
• static StringArray MP_ (discreteUncSetStrVars)
• static StringArray MP_ (discreteUncSetRealVars)
• static StringArray MP_ (histogramPointStrUncVars)
• static StringArray MP_ (linearEqScaleTypes)
• static StringArray MP_ (linearIneqScaleTypes)
• static StringArray VP_ (hpsa)
• static StringArray VP_ (ddss)
• static StringArray VP (duss)
• static StringArray VP (dsss)
• static BitArray MP (discreteDesignSetIntCat)
• static BitArray MP (discreteDesignSetRealCat)
• static BitArray MP (discreteStateSetIntCat)
• static BitArray MP (discreteStateSetRealCat)
• static BitArray MP (discreteUncSetIntCat)
• static BitArray MP (discreteUncSetRealCat)
• static Var brv MP2s (betaUncAlphas, 0.)
• static Var brv MP2s (betaUncBetas, 0.)
• static Var brv MP2s (exponentialUncBetas, 0.)
• static Var brv MP2s (exponentialUncVars, 0.)
• static Var brv MP2s (frechetUncAlphas, 2.)
• static Var brv MP2s (gammaUncAlphas, 0.)
• static Var brv MP2s (gammaUncBetas, 0.)
• static Var brv MP2s (gammaUncVars, 0.)
• static Var brv MP2s (gumbelUncAlphas, 0.)
• static Var brv MP2s (lognormalUncErrFacts, 1.)
• static Var brv MP2s (lognormalUncLambdas, 0.)
• static Var brv MP2s (lognormalUncLowerBnds, 0.)
• static Var brv MP2s (lognormalUncMeans, 0.)
• static Var brv MP2s (lognormalUncStdDevs, 0.)
• static Var brv MP2s (lognormalUncUpperBnds, std::numeric_limits< Real >::infinity())
• static Var brv MP2s (lognormalUncVars, 0.)
• static Var brv MP2s (lognormalUncZetas, 0.)
• static Var brv MP2s (loguniformUncLowerBnds, std::numeric_limits< Real >::infinity())
• static Var brv MP2s (loguniformUncUpperBnds, std::numeric_limits< Real >::infinity())
• static Var brv MP2s (loguniformUncVars, 0.)
• static Var brv MP2s (normalUncStdDevs, 0.)
• static Var brv MP2s (poissonUncLambdas, 0.)
• static Var brv MP2s (triangularUncLowerBnds, std::numeric_limits< Real >::infinity())
• static Var brv MP2s (triangularUncUpperBnds, std::numeric_limits< Real >::infinity())
• static Var brv MP2s (uniformUncLowerBnds, std::numeric_limits< Real >::infinity())
• static Var brv MP2s (uniformUncUpperBnds, std::numeric_limits< Real >::infinity())
• static Var brv MP2s (weibullUncAlphas, 0.)
• static Var brv MP2s (weibullUncBetas, 0.)
• static Var brv MP2s (binomialUncNumTrials, 0)
• static Var brv MP2s (binomialUncVars, 0)
• static Var brv MP2s (geometricUncVars, 0)
• static Var brv MP2s (hyperGeomUncNumDrawn, 0)
• static Var brv MP2s (hyperGeomUncSelectedPop, 0)
• static Var brv MP2s (hyperGeomUncTotalPop, 0)
• static Var brv MP2s (hyperGeomUncVars, 0)
• static Var brv MP2s (negBinomialUncNumTrials, 0)
• static Var brv MP2s (negBinomialUncVars, 0)
• static Var brv MP2s (poissonUncVars, 0)
void flatten_num_array (const std::vector< ContainerT > &input_array, IntArray **pia)

Free convenience function that flatten sizes of an array of std containers; takes an array of containers and returns an IntArray containing the sizes of each container in the input array. Note: Did not specialize for vector<RealVector> as no current use cases.

void dn2f_ (int *n, int *p, Real *x, Calcrj, int *iv, int *liv, int *lv, Real *v, int *ui, void *ur, Vf)
void dn2fb_ (int *n, int *p, Real *x, Real *b, Calcrj, int *iv, int *liv, int *lv, Real *v, int *ui, void *ur, Vf)
void dn2g_ (int *n, int *p, Real *x, Calcrj, Calcrj, int *iv, int *liv, int *lv, Real *v, int *ui, void *ur, Vf)
void divset_ (int *, int *, int *, Real *)
double dr7mdc_ (int *)
static void Rswapchk (Nl2Misc *q)
static int hasnaninf (const double *d, int n)
NLPQLPOptimizer * new_NLPQLPOptimizer (ProblemDescDB &problem_db, Model &model)
NLPQLPOptimizer * new_NLPQLPOptimizer (Model &model)
NOWPACOptimizer * new_NOWPACOptimizer (ProblemDescDB &problem_db, Model &model)
NOWPACOptimizer * new_NOWPACOptimizer (Model &model)
NPSOLOptimizer * new_NPSOLOptimizer (ProblemDescDB &problem_db)
NPSOLOptimizer * new_NPSOLOptimizer1 (Model &model)
NPSOLOptimizer * new_NPSOLOptimizer2 (Model &model, const int &derivative_level, const Real &conv_tol)
NPSOLOptimizer * new_NPSOLOptimizer3 (const RealVector &initial_point, const RealVector &var_, lower_bnds, const RealVector &var_upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_lower_bnds, const RealVector &lin_ineq_upper_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nonlin_ineq_lower_bnds, const RealVector &nonlin_ineq_upper_bnds, const RealVector &nonlin_eq_targets, void(*user_obj_eval)(int &, int &, double *, double &, double *, int &), void(*user_con_eval)(int &, int &, int &, int &, int *, double *, double *, double *, int &), const int &derivative_level, const Real &conv_tol)
NPSOLOptimizer * new_NPSOLOptimizer (ProblemDescDB &problem_db, Model &model)
NPSOLOptimizer * new_NPSOLOptimizer (Model &model)
NPSOLOptimizer * new_NPSOLOptimizer (Model &model, const int &, const Real &)
NPSOLOptimizer * new_NPSOLOptimizer (const RealVector &initial_point, const RealVector &var_, lower_bnds, const RealVector &var_upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_lower_bnds, const RealVector &lin_ineq_upper_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nonlin_ineq_lower_bnds, const RealVector &nonlin_ineq_upper_bnds, const RealVector &nonlin_eq_targets, void(*user_obj_eval)(int &, int &, double *, double &, double *, int &), void(*user_con_eval)(int &, int &, int &, int &, int *, double *, double *, double *, int &), const int &derivative_level, const Real &conv_tol)
12.1. DAKOTA NAMESPACE REFERENCE

- void `start_dakota_heartbeat` (int)
- void `dak_sighup` (int sig)
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ParallelLevel &pl)`
  *`MPIUnpackBuffer` extraction operator for `ParallelLevel`. Calls `read(MPIUnpackBuffer&)`.*
- `MPIPackBuffer & operator<<(MPIPackBuffer &s, const ParallelLevel &pl)`
  *`MPIPackBuffer` insertion operator for `ParallelLevel`. Calls `write(MPIPackBuffer&)`.
- std::istream & `operator>>(std::istream &s, ParamResponsePair &pair)`
  *`std::istream` extraction operator for `ParamResponsePair`*
- std::ostream & `operator<<(std::ostream &s, const ParamResponsePair &pair)`
  *`std::ostream` insertion operator for `ParamResponsePair`*
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ProgramOptions &progopt)`
  *`MPIUnpackBuffer` extraction operator.*
- `MPIPackBuffer & operator<<(MPIPackBuffer &s, const ProgramOptions &progopt)`
  *`MPIPackBuffer` insertion operator.*
- bool `operator==(const ParamResponsePair &pair1, const ParamResponsePair &pair2)`
  *equality operator for `ParamResponsePair`*
- bool `operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)`
  *inequality operator for `ParamResponsePair`*
- static void * `binsearch` (void *kw, size_t kwsize, size_t n, const char *key)
- static const char * `Begins` (const String &entry_name, const char *s)
- static void `Bad_name` (String entry_name, const char *where)
- static void `Locked_db` ()
- static void `Null_rep` (const char *who)
- static void `Null_rep1` (const char *who)
- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ProgramOptions &progopt)`
  *`MPIUnpackBuffer` extraction operator.*
- `MPIPackBuffer & operator<<(MPIPackBuffer &s, const ProgramOptions &progopt)`
  *`MPIPackBuffer` insertion operator.*
- bool `set_compare` (const ParamResponsePair &database_pr, const ActiveSet &search_set)
  *search function for a particular `ParamResponsePair` within a PRPList based on `ActiveSet` content (request vector and derivative variables vector)*
- bool `id_vars exact compare` (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr)
  *search function for a particular `ParamResponsePair` within a PRPMultiIndex*
- std::size_t `hash_value` (const ParamResponsePair &prp)
  *`hash_value` for `ParamResponsePairs` stored in a PRPMultiIndex*
- PRPCacheHIter `hashedCacheBegin` (PRPCache &prp cache)
  *hashed definition of cache begin*
- PRPCacheHIter `hashedCacheEnd` (PRPCache &prp cache)
  *hashed definition of cache end*
- PRPQueueHIter `hashedQueueBegin` (PRPQueue &prp queue)
  *hashed definition of queue begin*
- PRPQueueHIter `hashedQueueEnd` (PRPQueue &prp queue)
  *hashed definition of queue end*
• PRPCacheHIter lookup_by_val (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr)
  find a ParamResponsePair based on the interface id, variables, and ActiveSet search data within search_pr.
• PRPCacheHIter lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)
  find a ParamResponsePair within a PRPMultiIndexCache based on the interface id, variables, and ActiveSet search data
• PRPCacheOIter lookup_by_nearly_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, Real tol)
• PRPCacheOIter lookup_by_ids (PRPMultiIndexCache &prp_cache, const IntStringPair &search_ids)
  find a ParamResponsePair within a PRPMultiIndexCache based on search_ids (i.e. std::pair<eval_id,interface_id>) search data
• PRPCacheOIter lookup_by_ids (PRPMultiIndexCache &prp_cache, const IntStringPair &search_ids, const ParamResponsePair &search_pr)
• PRPQueueHIter lookup_by_val (PRPMultiIndexQueue &prp_queue, const ParamResponsePair &search_pr)
  find a ParamResponsePair based on the interface id, variables, and ActiveSet search data within search_pr.
• PRPQueueHIter lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)
  find a ParamResponsePair within a PRPMultiIndexQueue based on interface id, variables, and ActiveSet search data
• PRPQueueOIter lookup_by_eval_id (PRPMultiIndexQueue &prp_queue, int search_id)
  find a ParamResponsePair within a PRPMultiIndexQueue based on search_id (i.e. integer eval_id) search data
• void print_usage (std::ostream &s)
  print restart utility help message
• void print_restart (StringArray pos_args, String print_dest)
  print a restart file
• void print_restart_pdb (StringArray pos_args, String print_dest)
  print a restart file (PDB format)
• void print_restart_tabular (StringArray pos_args, String print_dest, unsigned short tabular_format, int tabular_precision)
  print a restart file (tabular format)
• void read_neutral (StringArray pos_args)
  read a restart file (neutral file format)
• void repair_restart (StringArray pos_args, String identifier_type)
  repair a restart file by removing corrupted evaluations
• void concatenate_restart (StringArray pos_args)
  concatenate multiple restart files
• static HANDLE * wait_setup (std::map< pid_t, int > *M, size_t *pn)
• static int wait_for_one (size_t n, HANDLE *h, int req1, size_t *pi)
• void gauss_legendre_pts_wts_1D (int level, RealVector &result_0, RealVector &result_1)
• void lagrange_interpolation_1d (const RealVector &samples, const RealVector &abscissa, const RealVector &values, RealVector &result)
• void kronecker_product_2d (const RealMatrix &matrix1, const RealMatrix &matrix2, RealMatrix &matrix)
• void get_chebyshev_points (int order, RealVector &points)
• void chebyshev_derivative_matrix (int order, RealMatrix &derivative_matrix, RealVector &points)
12.1. DAKOTA NAMESPACE REFERENCE

- `int salinas_main (int argc, char *argv[], MPI_Comm *comm)`
  subroutine interface to SALINAS simulation code
- `std::string get_cwd_str()`
- `std::vector<std::string> get_pathext()`
- `bool contains(const bfs::path &dir_path, const std::string &file_name, boost::filesystem::path &complete_path)`

Variables

- `PRPCache data_pairs`
  contains all parameter/response pairs.
- `double PI = boost::math::constants::pi<double>()`
  constant pi
- `double HALF_LOG_2PI = std::log(2.0*PI)/2.0`
  constant log(2*pi)/2.0
- `short abort_mode = ABORT_EXITS` by default Dakota exits or calls MPI_Abort on errors
- `std::ostream * dakota_cout = &std::cout`
  DAKOTA stdout initially points to `<std::cout`, but may be redirected to a tagged ofstream if there are `< concurrent iterators.
- `std::ostream * dakota_cerr = &std::cerr`
  DAKOTA stderr initially points to `<std::cerr`, but may be redirected to a tagged ofstream if there are `< concurrent iterators.
- `ResultsManager iterator_results_db`
  Global results database for iterator results.
- `int write_precision = 10`
  used in ostream data output functions `< (restart_util.cpp overrides default value)
- `MPIManager dummy_mpi_mgr`
  dummy MPIManager for ref initialization
- `ProgramOptions dummy_prg_opt`
  dummy ProgramOptions for ref initialization
- `OutputManager dummy_out_mgr`
  dummy OutputManager for ref initialization
- `ParallelLibrary dummy_lib`
  dummy ParallelLibrary for ref initialization
- `ProblemDescDB dummy_db`
  dummy ProblemDescDB for ref initialization
- `int mc_ptr_int = 0`
  global pointer for ModelCenter API
- `int dc_ptr_int = 0`
  global pointer for ModelCenter eval DB
- `ProblemDescDB * Dak_pddb`
  set by ProblemDescDB, for use in parsing
- `const size_t NPOS = ~(size_t)0`
special value returned by index() when entry not found

* const double **BIG_REAL_BOUND** = 1.0e+30
  bound beyond which constraints are considered inactive

* Interface dummy_interface
  dummy Interface object used for mandatory
  < reference initialization or default virtual < function return by reference when a real < Interface instance is unavailable

* Model dummy_model
  dummy Model object used for mandatory reference
  < initialization or default virtual function < return by reference when a real Model instance < is unavailable

* Iterator dummy_iterator
  dummy Iterator object used for mandatory
  < reference initialization or default virtual < function return by reference when a real Iterator instance is unavailable

* Dakota_func ∗ **DF**
* Dakota_func ∗ DakFuncs0
  const char ∗ **FIELD_NAMES** [ ]

* const int **NUMBER_OF_FIELDS** = 23

* static const int MPI_COMM_WORLD = 1
* static const int MPI_COMM_NULL = 0
* static const int MPI_COMM_SELF = 92
* static const int MPI_ANY_TAG = -1
* static void ∗ **MPI_REQUEST_NULL** = NULL
  FILE ∗ **nidrin**
  const size_t **NIDR_MAX_ERROR_LEN** = 8192
  maximum error length is roughly 100 lines at 80 char; using fixed error length instead of investing in converting to vsnprintf (C++11)

* static const char ∗ **aln_scaletypes** [ ] = { "auto", "log", "none", 0 }

* static Var_uinfo **CAUVLbl** [CAUVar_Nkinds]
* static Var_uinfo **DAUUVlbl** [DAUUIVar_Nkinds]
* static Var_uinfo **DAUSVlbl** [DAUSVar_Nkinds]
* static Var_uinfo **DAURVlbl** [DAURVar_Nkinds]
* static Var_uinfo **CEUVlbl** [CEUVar_Nkinds]
* static Var_uinfo **DEUUVlbl** [DEUUIVar_Nkinds]
* static Var_uinfo **DEUSVlbl** [DEUSVar_Nkinds]
* static Var_uinfo **DEURVlbl** [DEURVar_Nkinds]
* static Var_uinfo **DiscSetLbl** [DiscSetVar_Nkinds]

  Variables label array designations for design and state. All non-uncertain variables need to be in this array. Used
  in check_variables_node to check lengths and make_variable_defaults to build labels.

* static VLreal **VLUncertainReal** [NUM_UNC_REAL_CONT]

  Variables labels/bounds/values check array for real-valued uncertain variables; one array entry per contiguous
  container. These associate the individual variables given by, e.g., CAUVlbl, with the contiguous container in
  which they are stored.

* static VLint **VLUncertainInt** [NUM_UNC_INT_CONT]
Variables labels/bounds/values check array for integer-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUIVLbl, with the contiguous container in which they are stored.

- static VList VLUncertainStr [NUM_UNC_STR_CONT]

Variables labels/bounds/values check array for string-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUSVLbl, with the contiguous container in which they are stored.

- static int VLR_aleatory [NUM_UNC_REAL_CONT] = 
  1, 0, 1, 0 which uncertain real check array containers are aleatory (true = 1)
- static int VLI_aleatory [NUM_UNC_INT_CONT] = 
  1, 0 which uncertain integer check array containers are aleatory (true = 1)
- static int VLS_aleatory [NUM_UNC_STR_CONT] = 
  1, 0 which uncertain string check array containers are aleatory (true = 1)

- static Var check var_mp_check_cv []
- static Var check var_mp_check_dset []
- static Var check var_mp_check_cau []
- static Var check var_mp_check_daui []
- static Var check var_mp_check_daus []
- static Var check var_mp_check_daur []
- static Var check var_mp_check_ceu []
- static Var check var_mp_check_deui []
- static Var check var_mp_check_deus []
- static Var check var_mp_check_deur []
- static Var rcheck var_mp_cbound []

This is used within check variables node(): Var_RealBoundIPCheck() is applied to validate bounds and initial points.

- static Var icheck var_mp_drange []

This is used in check_variables_node(): Var_IntBoundIPCheck() is applied to validate bounds and initial points, and in make_variable_defaults(): Vgen_* is called to infer bounds.

- static time * start_time
- const char * SCIFIELD_NAMES []
- const int SCI_NUMBER_OF_FIELDS = 26
- const int LARGE_SCALE = 100
  a (perhaps arbitrary) definition of large scale; choose a large-scale algorithm if numVars >= LARGE_SCALE
- const double POW_VAL = 1.0
  offset used text book exponent: 1.0 is nominal, 1.4 used for B&B testing
- const String LEV_REF = "Dakota"
  levenshtein_distance computes the distance between its argument and this

12.1.1 Detailed Description

The primary namespace for DAKOTA. The Dakota namespace encapsulates the core classes of the DAKOTA framework and prevents name clashes with third-party libraries from methods and packages. The C++ source files defining these core classes reside in Dakota/src as *.cpp.

- Work directory TODO
  Doc: we will search for drivers in PATH, workdir (.), RUNDIR
  Remove legacy utilities (once concepts migrated)
In general review cases with race conditions such as single dir getting created / removed for each eval.

- Verify creation/removal in parallel runs (eval_comm_rank 0?); are there scenarios where we should create once ahead of time?
- Enforce tagging when async is possible
- Challenge of shared vs. distinct filesystems
- Verify template files exist at parse and that workdir parent exists
- Verify behavior when directory exists
- Allow recursive copy to descend to overwrite leaf nodes when directories already exist
- Old code setting permissions mask 0700
- Workdir with multiple analysis components (per analysis)
- Workdirs shared for each unique concurrent eval (not per eval ID)
- Evaluate environment variables
- How to manage drivers that want just param names in the work dir?!? I think arg_adjust is removing the directory args that aren’t needed.
- Consider making the class members for directories and files bfs::paths
- Behavior of file_save when directory not saved
- Error checking: directory was created directory path is a directory directory has rwx for user population worked try/catch around all fs operations
- Verify correct handling of relative vs. absolute files/dirs
- Enforce that first argument must be an executable program for all drivers; at least for fork
- Historical behaviors / features to consider Template dirs on PATH: likely no longer
  - Allowed FOO=zorch and would set that in the environment; could allow separate env var specification; otherwise likely remove
- TODO: pass environment to exec as separate pointer

**TESTING NEEDS**

- Allow nested quotes in driver, at least one level: `analysis_driver = 'ad.sh "-opt foo -opt1 goo"'` p.in.1 r.out.1
- Env vars will be carried along for now, not expanded before eval; set some helpful env vars before the eval.

### 12.1.2 Typedef Documentation

```cpp
typedef bmi::multi_index_container<Dakota::ParamResponsePair, bmi::indexed_by<bmi::ordered_non_unique<bmi::tag<bmi::ordered>, bmi::const_mem_fun<Dakota::ParamResponsePair, const IntStringPair&>, Dakota::ParamResponsePair::eval_interface_ids>>, bmi::hashed_non_unique<bmi::tag<bmi::hashed>>, bmi::identity<Dakota::ParamResponsePair>, partial_prp_hash, partial_prp_equality>> PRPMultiIndexCache
```

Boost Multi-Index Container for globally caching ParamResponsePairs.

For a global cache, both evaluation and interface id’s are used for tagging `ParamResponsePair` records.
typedef bmi::multi_index_container<Dakota::ParamResponsePair, bmi::indexed_by<bmi::ordered_unique<bmi::tag<bmi::ordered>, bmi::const_mem_fun<Dakota::ParamResponsePair, int, &Dakota::ParamResponsePair::eval>, bmi::hashed_non_unique<bmi::tag<bmi::hashed>, bmi::identity<Dakota::ParamResponsePair>>, partial_prp_hash, partial_prp_equality>>, bmi::hashed_non_unique<bmi::tag<bmi::hashed>, bmi::identity<Dakota::ParamResponsePair>>, partial_prp_hash, partial_prp_equality>>, PRPMultiIndexQueue

Boost Multi-Index Container for locally queueing ParamResponsePairs.

For a local queue, interface id’s are expected to be consistent, such that evaluation id’s are sufficient for tracking particular evaluations.

12.1.3 Enumeration Type Documentation

anonymous enum

Sub-methods, including sampling, inference algorithm, opt algorithm types.

Enumerator

SUBMETHOD COLLABORATIVE Type of hybrid meta-iterator:

12.1.4 Function Documentation

CommandShell & flush ( CommandShell & shell )

convenient shell manipulator function to “flush” the shell

global convenience function for manipulating the shell; invokes the class member flush function.

References CommandShell::flush().

Referenced by SysCallApplicInterface::spawn_analysis_to_shell(), SysCallApplicInterface::spawn_evaluation_to_shell(), SysCallApplicInterface::spawn_input_filter_to_shell(), and SysCallApplicInterface::spawn_output_filter_to_shell().

void abort_throw_or_exit ( int dakota_code )

throw or exit depending on abort_mode

Throw a Boost system_error or call std::exit, with (256 + dakota_code), where dakota_code < 0

RATIONALE: Avoid common ”standard” exit codes and signals (signum.h) as well as uncaught signals / uncatchable SIGKILL which return 128

- <signum> on Linux = [129, 192]

Return a value in [0,255] since some operating systems only return the 8 least significant bits, leaves [193, 255] for Dakota. This should make return codes consistent cross-platform.

References abort_mode.

Referenced by abort_handler(), and ParallelLibrary::abort_helper().

void register_signal_handlers ( )

Tie various signal handlers to Dakota’s abort_handler function.

Global function to register signal handlers at top-level.

References abort_handler().

Referenced by main().
void mpi_debug_hold ( )

Global function to hold Dakota processes to help with MPI debugging.

See details in code for details, depending on MPI implementation in use.

Referenced by main().

T Dakota::abort_handler_t ( int code )

Templatized abort_handler_t method that allows for convenient return from methods that otherwise have no sensible return from error clauses. Usage: MyType& method() { return abort_handler<MyType&>(-1); }

References abort_handler().

void svd ( RealMatrix & matrix, RealVector & singular_vals, RealMatrix & v_trans, bool compute_vectors = true )

Compute the SVD of an arbitrary matrix A = USV^T.

Uses Teuchos::LAPACK.GESVD() to compute the singular value decomposition, overwriting A with the left singular vectors U (or destroying A if compute_vectors = false); optionally returns right singular vectors in v_trans.

References abort_handler().

Referenced by NonDBayesCalibration::build_model_discrepancy(), PebblBranchSub::candidateSolution(), NestedModel::ccv_index_map(), NestedModel::cdv_index_map(), NestedModel::cdsv_index_map(), ActiveSubspaceModel::compute_svd(), ActiveSubspaceModel::computeBingLiCriterion(), ActiveSubspaceModel::computeConstantineMetric(), Variables::continuous_variable_id(), Variables::continuous_variable_ids(), Variables::continuous_variable_label(), Variables::continuous_variable_labels(), Variables::continuous_variable_type(), Variables::continuous_variable_types(), SharedVariablesData::copy(), NestedModel::cv_index_map(), Model::discrete_int_sets(), Variables::discrete_int_variable_label(), Variables::discrete_int_variable_labels(), Variables::discrete_int_variable_types(), Variables::discrete_real_variable_label(), Variables::discrete_real_variable_labels(), Variables::discrete_real_variable_types(), Variables::discrete_real_variable_values(), Variables::discrete_set_int_values(), Model::discrete_set_real_values(), Variables::discrete_string_variable_label(), Variables::discrete_string_variable_labels(), Variables::discrete_string_variable_types(), Variables::discrete_string_variable_values(), Variables::discrete_string_variable_types(), ParamStudy::distribute(), NestedModel::div_index_map(), NestedModel::dry_index_map(), NestedModel::dsv_index_map(), DataTransformModel::get_hyperparam_vc_index(), Variables::inactive_continuous_variable_id(), Variables::inactive_continuous_variable_labels(), Variables::inactive_continuous_variable_types(), Variables::inactive_continuous_variable_values(), Variables::inactive_discrete_int_variable_label(), Variables::inactive_discrete_int_variable_labels(), Variables::inactive_discrete_int_variable_types(), Variables::inactive_discrete_real_variable_label(), Variables::inactive_discrete_real_variable_labels(), Variables::inactive_discrete_real_variable_types(), Variables::inactive_discrete_string_variable_label(), Variables::inactive_discrete_string_variable_labels(), Variables::inactive_discrete_string_variable_types(), DataTransformModel::init_continuous_vars(), RecastModel::init_variables(), NonD::NonD(), Optimizer::Optimizer(), SensAnalysisGlobal::partial_corr(), ParamStudy::pre_run(), ProbabilityTransformModel::ProbabilityTransformModel(), Constraints::reshape(), NonD::resize(), singular_values(), NonD::size(), PebblBranchSub::splitComputation(), NonD::transform_model(), ReducedBasis::update_svd(), DataTransformModel::variables_expand(), RandomFieldModel::variables_resize(), AdaptedBasisModel::variables_resize(), and ActiveSubspaceModel::variables_resize().

int qr ( RealMatrix & A )

Compute an in-place QR factorization A = QR.

Uses Teuchos::LAPACK.GEQRF() to compute the QR decomposition, overwriting A with the transformations and R.

References abort_handler().

Referenced by SensAnalysisGlobal::partial_corr().
12.1. DAKOTA NAMESPACE REFERENCE

```cpp
int qr_rsolve ( const RealMatrix & q, bool transpose, RealMatrix & rhs )
```

Perform a multiple right-hand sides $R\inv \times \text{rhs}$ solve using the $R$ from a qr factorization.

- Returns info $> 0$ if the matrix is singular
- Uses Teuchos::LAPACK::TRTRS() to perform a triangular backsolve
- References abort_handler().
- Referenced by SensAnalysisGlobal::partial_corr().

```cpp
bool Dakota::operator!=( const ActiveSet & set1, const ActiveSet & set2 ) [inline]
```

inequality operator for ActiveSet

inequality operator

```cpp
bool Dakota::operator==( const Model & m1, const Model & m2 ) [inline]
```

equality operator for Envelope is true if same letter instance

- equality operator (detect same letter instance)
- References Model::modelRep.

```cpp
bool Dakota::operator!=( const Model & m1, const Model & m2 ) [inline]
```

inequality operator for Envelope is true if different letter instance

- inequality operator (detect different letter instances)
- References Model::modelRep.

```cpp
bool Dakota::get_bounds ( const RealVector & lower_source, const RealVector & upper_source, Real big_bound_size, Real no_value, VecT & lower_target, VecT & upper_target )
```

Adapter for copying continuous variables data from Dakota RealVector into TPL vectors

- Referenced by get_linear_constraints(), and get_variable_bounds().

```cpp
void Dakota::get_bounds ( const SetT & source_set, VecT & lower_target, VecT & upper_target, int target_offset )
```

Adapter originating from (and somewhat specialized based on) APPSOptimizer for copying discrete variables from a set-based Dakota container into TPL vectors

```cpp
bool Dakota::get_mixed_bounds ( const MaskType & mask_set, const SetArray & source_set,
const Teuchos::SerialDenseVector< OrdinalType, ScalarType > & lower_source, const
Teuchos::SerialDenseVector< OrdinalType, ScalarType > & upper_source, VectorType2 & lower_target,
VectorType2 & upper_target, ScalarType bigBoundSize, ScalarType no_value, int target_offset = 0 )
```

Adapter originating from (and somewhat specialized based on) APPSOptimizer for copying discrete integer variables data with bit masking from Dakota into TPL vectors

- Referenced by get_variable_bounds().
bool Dakota::get_variable_bounds ( Model & model, Real big_real_bound_size, int big_int_bound_size, typename AdapterT::VecT & lower, typename AdapterT::VecT & upper )

Adapter originating from (and somewhat specialized based on) APPSOptimizer for copying heterogeneous bounded data from Dakota::Variables into concatenated TPL vectors

References Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::cv(), Model::discrete_int_lower_bounds(), Model::discrete_int_sets(), Model::discrete_int_upper_bounds(), Model::discrete_real_lower_bounds(), Model::discrete_real_upper_bounds(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::div(), Model::drv(), get_bounds(), and get_mixed_bounds().

int Dakota::configure_inequality_constraint_maps ( const Model & model, Real big_real_bound_size, CONSTRAINT_TYPE ctype, IVecT & map_indices, RVecT & map_multipliers, RVecT & map_offsets, Real scaling = 1.0 )

Adapter for configuring inequality constraint maps used when transferring data between Dakota and a TPL

References Model::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_linear_ineq_constraints(), and Model::num_nonlinear_ineq_constraints().

Referenced by Optimizer::configure_constraint_maps().

void Dakota::configure_equality_constraint_maps ( Model & model, CONSTRAINT_TYPE ctype, IVecT & indices, size_t index_offset, RVecT & multipliers, RVecT & values, bool make_one_sided )

Adapter for configuring equality constraint maps used when transferring data between Dakota and a TPL

References Model::linear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Model::num_linear_eq_constraints(), and Model::num_nonlinear_eq_constraints().

void Dakota::get_linear_constraints ( Model & model, Real big_real_bound_size, typename AdapterT::VecT & lin_ineq_lower_bnds, typename AdapterT::VecT & lin_ineq_upper_bnds, typename AdapterT::VecT & lin_eq_targets, typename AdapterT::MatT & lin_ineq_coeffs, typename AdapterT::MatT & lin_eq_coeffs )

Adapter based initially on APPSOptimizer for linear constraint maps and including matrix and bounds data; bundles a few steps together which could (should?) be broken into two or more adapters

References copy_data(), get_bounds(), Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), and Model::linear_ineq_constraint_upper_bounds().

void Dakota::set_best_responses ( typename AdapterT::OptT & optimizer, const Model & model, const std::vector< int > constraint_map_indices, const std::vector< double > constraint_map_multipliers, const std::vector< double > constraint_map_offsets, ResponseArray & response_array )

Data adapter for use by third-party opt packages to transfer response data to Dakota

References Model::num_functions(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), and Model::primary_response_fn_sense().

void Dakota::set_variables ( const VectorType & source, Model & model, Variables & vars )

cpy appropriate slices of source vector to Dakota::Variables

References Variables::continuous_variables(), copy_data(), copy_data_partial(), Variables::cv(), Model::discrete_int_sets(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Model::discrete_set_int_values(),

CHAPTER 12. NAMESPACE DOCUMENTATION
12.1. DAKOTA NAMESPACE REFERENCE

Model::discrete_set_real_values(), Model::discrete_set_string_values(), Variables::discrete_string_variable(), Variables::div(), Variables::drv(), Variables::dsv(), and set_index_to_value().

void Dakota::get_variables ( Model & model, VectorType & vec )

copy the various pieces comprising Dakota::Variables into a concatenated TPL vector

References abort_handler(), Model::continuous_variables(), copy_data(), Model::cv(), Model::discrete_int_sets(), Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::discrete_variables(), Model::div(), Model::drv(), and Model::dsv().

void Dakota::get_responses ( const Model & model, const RealVector & dak_fn_vals, const std::vector<int> constraint_map_indices, const std::vector<double> constraint_map_multipliers, const std::vector<double> constraint_map_offsets, vectorType & f_vec, vectorType & cEqs_vec, vectorType & clEqs_vec )

Data adapter to transfer data from Dakota to third-party opt packages

References Model::num_nonlinear_eq_constraints(), and Model::primary_response_fn_sense().

Referenced by Optimizer::get_responses_from_dakota().

void Dakota::get_nonlinear_eq_constraints ( Model & model, const RVecT & curr_resp_vals, RVecT & values )

Data adapter to transfer data from Dakota to third-party opt packages

References Model::nonlinear_eq_constraint_targets(), and Model::num_nonlinear_eq_constraints().

void Dakota::get_nonlinear_eq_constraints ( Model & model, const RealVector & curr_resp_vals, RVecT & values, Real scale, int offset = 0 )

Data adapter to transfer data from Dakota to third-party opt packages

References Model::nonlinear_eq_constraint_targets(), and Model::num_nonlinear_eq_constraints().

void Dakota::get_nonlinear_constraints ( Model & model, VecT & nonlin_ineq_lower, VecT & nonlin_ineq_upper, VecT & nonlin_eq_targets )

Would like to combine the previous adapter with this one (based on APPSOptimizer and COLINOptimizer) and then see how much more generalization is needed to support other TPLs like JEGA.

Data adapter to transfer data from Dakota to third-party opt packages

References copy_data(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), and Model::nonlinear_ineq_constraint_upper_bounds().

Referenced by COLINApplication::set_problem().

bool Dakota::operator!= ( const Response & resp1, const Response & resp2 ) [inline]

inequality operator for Response

inequality operator

bool Dakota::operator!= ( const Variables & vars1, const Variables & vars2 ) [inline]

inequality operator for Variables

strict inequality operator
void Dakota::write_ordered ( std::ostream & s, const SizetArray & comp_totals, const Teuchos::SerialDenseVector< OrdinalType, ScalarType1 > & c_vector, const Teuchos::SerialDenseVector< OrdinalType, ScalarType2 > & di_vector, const Teuchos::SerialDenseVector< OrdinalType, ScalarType3 > & ds_vector, const Teuchos::SerialDenseVector< OrdinalType, ScalarType4 > & dr_vector ) [inline]

free function to write Variables data vectors in input spec ordering
written for arbitrary types, but typical use will be ScalarType1 = Real, ScalarType2 = int, ScalarType3 = string, and ScalarType4 = int or Real.
Referenced by ParamStudy::pre_run().

void Dakota::write_ordered ( std::ostream & s, const SizetArray & comp_totals, const Teuchos::SerialDenseVector< OrdinalType, ScalarType1 > & c_vector, const Teuchos::SerialDenseVector< OrdinalType, ScalarType2 > & di_vector, const Teuchos::SerialDenseVector< OrdinalType, ScalarType3 > & ds_array, const Teuchos::SerialDenseVector< OrdinalType, ScalarType4 > & dr_vector ) [inline]

free function to write Variables data vectors in input spec ordering
written for arbitrary types, but typical use will be ScalarType1 = Real, ScalarType2 = int, ScalarType3 = string, and ScalarType4 = int or Real.

void copy_field_data ( const RealVector & fn_vals, RealMatrix & fn_grad, const RealSymMatrixArray & fn_hess, size_t offset, size_t num_fns, Response & response )

This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.
References Response::active_set_request_vector(), Response::function_gradient_view(), Response::function_hessian_view(), and Response::function_value().
Referenced by ExperimentData::scale_residuals().

void Dakota::copy_field_data ( const RealVector & fn_vals, RealMatrix & fn_grad, const RealSymMatrixArray & fn_hess, size_t offset, size_t num_fns, short total_asv, Response & response )

This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.
References Response::function_gradient_view(), Response::function_hessian_view(), and Response::function_value().

void symmetric_eigenvalue_decomposition ( const RealSymMatrix & matrix, RealVector & eigenvalues, RealMatrix & eigenvectors )

Computes the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.
Eigenvalues are returned in ascending order.
References symmetric_eigenvalue_decomposition().
Referenced by NonDBayesCalibration::get_positive_definite_covariance_from_hessian(), and symmetric_eigenvalue_decomposition().

Real Dakota::getdist ( const RealVector & x1, const RealVector & x2 )

Gets the Euclidean distance between x1 and x2
Referenced by mindist(), and mindistindx().
12.1. DAKOTA NAMESPACE REFERENCE

Real Dakota::mindist ( const RealVector & x, const RealMatrix & xset, int except )

Returns the minimum distance between the point x and the points in the set xset (compares against all points in xset except point "except"): if except is not needed, pass 0.
  References getdist().
  Referenced by getRmax().

Real Dakota::mindistindx ( const RealVector & x, const RealMatrix & xset, const IntArray & indx )

Gets the min distance between x and points in the set xset defined by the nindx values in indx.
  References getdist().
  Referenced by GaussProcApproximation::pointsel_add_sel().

Real Dakota::getRmax ( const RealMatrix & xset )

Gets the maximum of the min distance between each point and the rest of the set.
  References mindist().
  Referenced by GaussProcApproximation::pointsel_add_sel().

int Dakota::start_grid_computing ( char * analysis_driver_script, char * params_file, char * results_file )

sample function prototype for launching grid computing

int Dakota::stop_grid_computing ( )

sample function prototype for terminating grid computing

int Dakota::perform_analysis ( char * iteration_num )

sample function prototype for submitting a grid evaluation

string Dakota::asstring ( const T & val )

Creates a string from the argument val using an ostringstream.
  This only gets used in this file and is only ever called with ints so no error checking is in place.
  Parameters

| val | The value of type T to convert to a string. |

Returns
  The string representation of val created using an ostringstream.
  Referenced by JEGAOptimizer::LoadTheConstraints().

void start_dakota_heartbeat ( int seconds )

Heartbeat function provided by dakota_filesystem_utils; pass output interval in seconds, or -1 to use SDAKOTA HEARTBEAT
  Referenced by OutputManager::OutputManager().


**bool Dakota::operator==( const ParamResponsePair & \textit{pair1}, const ParamResponsePair & \textit{pair2} ) [inline]**

equality operator for \textit{ParamResponsePair}  

- equality operator  

  References ParamResponsePair::evalInterfaceIds, ParamResponsePair::prpResponse, and ParamResponsePair-::prpVariables.

**bool Dakota::operator!=( const ParamResponsePair & \textit{pair1}, const ParamResponsePair & \textit{pair2} ) [inline]**

inequality operator for \textit{ParamResponsePair}  

- inequality operator

**bool Dakota::set_compare ( const ParamResponsePair & database-pr, const ActiveSet & search-set ) [inline]**

search function for a particular \textit{ParamResponsePair} within a PRPList based on \textit{ActiveSet} content (request vector and derivative variables vector)  

- a global function to compare the \textit{ActiveSet} of a particular database-pr (presumed to be in the global history list) with a passed in \textit{ActiveSet} (search-set).  

  References ParamResponsePair::active_set(), ActiveSet::derivative_vector(), and ActiveSet::request_vector().  

  Referenced by lookup_by_val().

**bool Dakota::id_vars_exact_compare ( const ParamResponsePair & database-pr, const ParamResponsePair & search-pr ) [inline]**

search function for a particular \textit{ParamResponsePair} within a PRPMultiIndex  

- a global function to compare the interface id and variables of a particular database-pr (presumed to be in the global history list) with a passed key of interface id and variables provided by search-pr.  

  References ParamResponsePair::interface_id(), and ParamResponsePair::variables().  

  Referenced by partial_prp_equality::operator().

**PRPCacheHIter Dakota::lookup_by_val ( PRPMultiIndexCache & prp_cache, const ParamResponsePair & search_pr ) [inline]**

find a \textit{ParamResponsePair} based on the interface id, variables, and \textit{ActiveSet} search data within search_pr.  

- Lookup occurs in two steps: (1) PRPMultiIndexCache lookup based on strict equality in interface id and variables, and (2) set_compare() post-processing based on \textit{ActiveSet} subset logic.  

  References ParamResponsePair::active_set(), and set_compare().  

  Referenced by NonDQUESTOBayesCalibration::aggregate_acceptance_chain(), NonDDREAMBayesCalibration-::archive_acceptance_chain(), Model::db_lookup(), ApplicationInterface::duplication_detect(), SurrBasedLocalMinimizer-::find_response(), Minimizer::local_recast_retrieve(), lookup_by_val(), SurrBasedMinimizer::print_results(), Least-Sq::print_results(), Optimizer::print_results(), DiscrepancyCorrection::search_db(), and NonDLocalReliability::update_mpp_search_data().

**PRPQueueHIter Dakota::lookup_by_val ( PRPMultiIndexQueue & prp_queue, const ParamResponsePair & search_pr ) [inline]**

find a \textit{ParamResponsePair} based on the interface id, variables, and \textit{ActiveSet} search data within search_pr.
Lookup occurs in two steps: (1) PRPMultiIndexQueue lookup based on strict equality in interface id and variables, and (2) set_compare() post-processing based on ActiveSet subset logic. References ParamResponsePair::active_set(), and set_compare().

```cpp
void print_restart ( StringArray pos_args, String print_dest )
print a restart file
Usage: ”dakota_restart_util print dakota.rst”
"dakota_restart_util to neutral dakota.rst dakota.neu”
Prints all evlas. in full precision to either stdout or a neutral file. The former is useful for ensuring that duplicate detection is successful in a restarted run (e.g., starting a new method from the previous best), and the latter is used for translating binary files between platforms.
References abort_handler(), ParamResponsePair::eval_id(), ParamResponsePair::write_annotated(), and write_precision.
Referenced by main().
```

```cpp
void print_restart_pdb ( StringArray pos_args, String print_dest )
print a restart file (PDB format)
Usage: ”dakota_restart_util to_pdb dakota.rst dakota.pdb”
Unrolls all data associated with a particular tag for all evaluations and then writes this data in a tabular format (e.g., to a PDB database or MATLAB/TECPLOT data file).
References abort_handler(), Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), and Response::function_values().
Referenced by main().
```

```cpp
void print_restart_tabular ( StringArray pos_args, String print_dest, unsigned short tabular_format, int tabular_precision )
print a restart file (tabular format)
Usage: ”dakota_restart_util to_tabular dakota.rst dakota.txt”
Unrolls all data associated with a particular tag for all evaluations and then writes this data in a tabular format (e.g., to a PDB database or MATLAB/TECPLOT data file).
References abort_handler(), Variables::acv(), Variables::adv(), Variables::adrv(), Variables::adsv(), Variables::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Variables::all_discrete_string_variable_labels(), Response::function_labels(), ParamResponsePair::interface_id(), ParamResponsePair::response(), ParamResponsePair::variables(), write_precision, ParamResponsePair::write_tabular(), and ParamResponsePair::write_tabular_labels().
Referenced by main().
```

```cpp
void read_neutral ( StringArray pos_args )
read a restart file (neutral file format)
Usage: ”dakota_restart_util from_neutral dakota.neu dakota.rst”
Reads evaluations from a neutral file. This is used for translating binary files between platforms.
References abort_handler(), and ParamResponsePair::read_annotated().
Referenced by main().
void repair_restart ( StringArray pos_args, String identifier_type )

repair a restart file by removing corrupted evaluations

Usage: "dakota_restart_util remove 0.0 dakota_old.rst dakota_new.rst"
"dakota_restart_util remove_ids 2 7 13 dakota_old.rst dakota_new.rst"

Repairs a restart file by removing corrupted evaluations. The identifier for evaluation removal can be either a double precision number (all evaluations having a matching response function value are removed) or a list of integers (all evaluations with matching evaluation ids are removed).

References abort_handler(), Response::active_set_request_vector(), contains(), ParamResponsePair::eval_id(), Response::function_values(), and ParamResponsePair::response().

Referenced by main().

void concatenate_restart ( StringArray pos_args )

concatenate multiple restart files

Usage: "dakota_restart_util cat dakota_1.rst ... dakota_n.rst dakota_new.rst"

Combines multiple restart files into a single restart database.

References abort_handler().

Referenced by main().

std::vector<std::string> Dakota::get_pathext ( )

Utility function for executable file search algorithms

Referenced by WorkdirHelper::which().

bool Dakota::contains ( const bfs::path & dir_path, const std::string & file_name, boost::filesystem::path & complete_filepath ) [inline]

Utility function for "which" sets complete_filepath from dir_path/file_name combo

12.1.5 Variable Documentation

short abort_mode = ABORT_EXITS

by default Dakota exits or calls MPL_Abort on errors

whether dakota exits/abort or throws on errors

Referenced by abort_throw_or_exit(), Environment::exit_mode(), and PythonInterface::python_run().

Dakota_funcs DakFuncs0

Initial value:

= {
  fprintf,
  abort_handler,
  dlsolver_option,
  continuous_lower_bounds1,
  continuous_upper_bounds1,
  nonlinear_ineq_constraint_lower_bounds1,
  nonlinear_ineq_constraint_upper_bounds1,
  nonlinear_eq_constraint Targets1,
  linear_ineq_constraint_lower_bounds1,
  linear_ineq_constraint_upper_bounds1,
  linear_eq_constraintTargets1,
  linear_ineq_constraint_coeffs1,
  linear_eq_constraint_coeffs1,
12.1. DAKOTA NAMESPACE REFERENCE

ComputeResponses1, GetFuncs1, GetGrads1, GetContVars1, SetBestContVars1, SetBestDiscVars1, SetBestRespFns1, Get_Real1, Get_Int1, Get_Bool1

const char* FIELD_NAMES[]

Initial value:

= { "numFns", "numVars", "numACV", "numADIV", "numADRV", "numDerivVars", "xC", "xDI", "xDR", "xCLabels", "xDILabels", "xDRLabels", "directFnASV", "directFnDVV", "fnFlag", "gradFlag", "hessFlag", "fnVals", "fnGrads", "fnHessians", "fnLabels", "failure", "currEvalId" }

fields to pass to Matlab in Dakota structure
Referred by MatlabInterface::matlab_engine_run(), and MatlabInterface::MatlabInterface().

const int NUMBER_OF_FIELDS = 23

number of fields in above structure
Referred by MatlabInterface::matlab_engine_run(), and MatlabInterface::MatlabInterface().

Var_uinfo CAUVLbl[CAUVar_Nkinds] [static]

Initial value:


Var_uinfo DAUIVLbl[DAUIVar_Nkinds] [static]

Initial value:

= { VarLabelInfo(puv_, PoissonUnc), VarLabelInfo(biuv_, BinomialUnc), VarLabelInfo(nbuv_, NegBinomialUnc), VarLabelInfo(euv_, GeometricUnc), VarLabelInfo(hguv_, HyperGeomUnc), VarLabelInfo(hpiuv_, HistogramPtIntUnc) }
Var_uinfo DAUSVLbl[DAUSVar_Nkinds] [static]
Initial value:

= {
   VarLabelInfo(hpsuv, HistogramPtStrUnc)
}

Var_uinfo DAURVLbl[DAURVar_Nkinds] [static]
Initial value:

= {
   VarLabelInfo(hpruv, HistogramPtRealUnc)
}

Var_uinfo CEUVLbl[CEUVar_Nkinds] [static]
Initial value:

= {
   VarLabelInfo(ciuv, ContinuousIntervalUnc)
}

Var_uinfo DEUIVLbl[DEUIVar_Nkinds] [static]
Initial value:

= {
   VarLabelInfo(diuv, DiscreteIntervalUnc),
   VarLabelInfo(dusiv, DiscreteUncSetInt)
}

Var_uinfo DEUSVLbl[DEUSVar_Nkinds] [static]
Initial value:

= {
   VarLabelInfo(dussv, DiscreteUncSetStr)
}

Var_uinfo DEURVLbl[DEURVar_Nkinds] [static]
Initial value:

= {
   VarLabelInfo(dussr, DiscreteUncSetReal)
}
12.1. DAKOTA NAMESPACE REFERENCE

Var_uinfo DiscSetLbl[DiscSetVar_Nkinds] [static]

Initial value:
= { VarLabelInfo(ddsi, DiscreteDesSetInt),
  VarLabelInfo(doss, DiscreteDesSetStr),
  VarLabelInfo(ddsr, DiscreteDesSetReal),
  VarLabelInfo(dsi, DiscreteStateSetInt),
  VarLabelInfo(di, DiscreteStateSetStr),
  VarLabelInfo(dsr, DiscreteStateSetReal) }

VarLabelChk DesignAndStateLabelsCheck[] [static]

Initial value:
= { AVI numContinuousDesVars, AVI continuousDesignLabels, "cdv", "cdv_descriptors" },
  { AVI numDiscreteDesRangeVars, AVI discreteDesignRangeLabels, "drv", "drv_descriptors" },
  { AVI numDiscreteDesSetIntVars, AVI discreteDesignSetIntLabels, "dsiv", "dsiv_descriptors" },
  { AVI numDiscreteDesSetStrVars, AVI discreteDesignSetStrLabels, "dssv", "dssv_descriptors" },
  { AVI numDiscreteDesSetRealVars, AVI discreteDesignSetRealLabels, "dssrv", "dssrv_descriptors" },
  { AVI numContinuousStateVars, AVI continuousStateLabels, "csv", "csv_descriptors" },
  { AVI numDiscreteStateRangeVars, AVI discreteStateRangeLabels, "dsrv", "dsrv_descriptors" },
  { AVI numDiscreteStateSetIntVars, AVI discreteStateSetIntLabels, "dssiv", "dssiv_descriptors" },
  { AVI numDiscreteStateSetStrVars, AVI discreteStateSetStrLabels, "dsssv", "dsssv_descriptors" },
  { AVI numDiscreteStateSetRealVars, AVI discreteStateSetRealLabels, "dssrv", "dssrv_descriptors" },
  { AVI numContinuousDesVars, AVI continuousDesignScaleTypes, 0, "cdv_scale_types" }

Variables label array designations for design and state. All non-uncertain variables need to be in this array. Used in check_variables_node to check lengths and make_variable_defaults to build labels.

Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().

VLreal VLUncertainReal[NUM_UNC_REAL_CONT] [static]

Initial value:
= { CAUVar_Nkinds, AVI CAUV, CAUVlbl,
    DVR continuousAleatoryUncLabels,
    DVR continuousAleatoryUncLowerBnds,
    DVR continuousAleatoryUncUpperBnds,
    DVR continuousAleatoryUncVars },
  { CEUVar_Nkinds, AVI CEUV, CEUVlbl,
    DVR continuousEpistemicUncLabels,
    DVR continuousEpistemicUncLowerBnds,
    DVR continuousEpistemicUncUpperBnds,
    DVR continuousEpistemicUncVars },
  { DAURVar_Nkinds, AVI DAURV, DAURVlbl,
    DVR discreteRealAleatoryUncLabels,
    DVR discreteRealAleatoryUncLowerBnds,
    DVR discreteRealAleatoryUncUpperBnds,
    DVR discreteRealAleatoryUncVars },
  { DEURVar_Nkinds, AVI DEURV, DEURVlbl,
    DVR discreteRealEpistemicUncLabels,
    DVR discreteRealEpistemicUncLowerBnds,
    DVR discreteRealEpistemicUncUpperBnds,
    DVR discreteRealEpistemicUncVars } }

Variables labels/bounds/values check array for real-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., CAUVlbl, with the contiguous container in which they are stored.

Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().
VLint VLUncertainInt[NUM_UNC_INT_CONT] [static]

Initial value:

= {
   {DAUIVar_Nkinds, AVI DAUIv, DAUIVLbl,
    DVR discreteIntAleatoryUncLabels,
    DVR discreteIntAleatoryUncLowerBnds,
    DVR discreteIntAleatoryUncUpperBnds,
    DVR discreteIntAleatoryUncVars},
   {DEUIVar_Nkinds, AVI DEUIv, DEUIVLbl,
    DVR discreteIntEpistemicUncLabels,
    DVR discreteIntEpistemicUncLowerBnds,
    DVR discreteIntEpistemicUncUpperBnds,
    DVR discreteIntEpistemicUncVars}}

Variables labels/bounds/values check array for integer-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUIVLbl, with the contiguous container in which they are stored.

Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().

VLstr VLUncertainStr[NUM_UNC_STR_CONT] [static]

Initial value:

= {
   {DAUSVar_Nkinds, AVI DAUSv, DAUSVLbl,
    DVR discreteStrAleatoryUncLabels,
    DVR discreteStrAleatoryUncLowerBnds,
    DVR discreteStrAleatoryUncUpperBnds,
    DVR discreteStrAleatoryUncVars},
   {DEUSVar_Nkinds, AVI DEUSv, DEUSVLbl,
    DVR discreteStrEpistemicUncLabels,
    DVR discreteStrEpistemicUncLowerBnds,
    DVR discreteStrEpistemicUncUpperBnds,
    DVR discreteStrEpistemicUncVars}}

Variables labels/bounds/values check array for string-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUSVLbl, with the contiguous container in which they are stored.

Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().

Var_check var_mp_check_cv[] [static]

Initial value:

= {
   Vchk_3(continuous,design,ContinuousDes),
   Vchk_3(continuous,state,ContinuousState) }

Var_check var_mp_check_dset[] [static]

Initial value:

= {
   Vchk_3(discrete_design_set_integer,DiscreteDesSetInt),
   Vchk_3(discrete_design_set_string,DiscreteDesSetStr),
   Vchk_3(discrete_design_set_real,DiscreteDesSetReal),
   Vchk_3(discrete_state_set_integer,DiscreteStateSetInt),
   Vchk_3(discrete_state_set_string,DiscreteStateSetStr),
   Vchk_3(discrete_state_set_real,DiscreteStateSetReal) }
Var_check var_mp_check_cau[] [static]

Initial value:

= { VchkJ(normal_uncertain, NormalUnc),
    VchkJ(lognormal_uncertain, LognormalUnc),
    VchkJ(uniform_uncertain, UniformUnc),
    VchkJ(loguniform_uncertain, LoguniformUnc),
    VchkJ(triangular_uncertain, TriangularUnc),
    VchkJ(exponential_uncertain, ExponentialUnc),
    VchkJ(beta_uncertain, BetaUnc),
    VchkJ(gamma_uncertain, GammaUnc),
    VchkJ(gumbel_uncertain, GumbelUnc),
    VchkJ(frechet_uncertain, FrechetUnc),
    VchkJ(weibull_uncertain, WeibullUnc),
    VchkJ(histogram_bin_uncertain, HistogramBinUnc) }

Var_check var_mp_check_daui[] [static]

Initial value:

= { VchkJ(poisson_uncertain, PoissonUnc),
    VchkJ(binomial_uncertain, BinomialUnc),
    VchkJ(negative_binomial_uncertain, NegBinomialUnc),
    VchkJ(geometric_uncertain, GeometricUnc),
    VchkJ(hypergeometric_uncertain, HyperGeomUnc),
    VchkJ(histogram_point_int_uncertain, HistogramPtIntUnc) }

Var_check var_mp_check_daus[] [static]

Initial value:

= { VchkJ(histogram_point_str_uncertain, HistogramPtStrUnc) }

Var_check var_mp_check_daur[] [static]

Initial value:

= { VchkJ(histogram_point_real_uncertain, HistogramPtRealUnc) }

Var_check var_mp_check_ceu[] [static]

Initial value:

= { VchkJ(continuous_interval_uncertain, ContinuousIntervalUnc) }

Var_check var_mp_check_deui[] [static]

Initial value:

= { VchkJ(discrete_interval_uncertain, DiscreteIntervalUnc),
    VchkJ(discrete_uncertain_set_integer, DiscreteUncSetInt) }
`Var_check var_mp_check_deus[] [static]`
Initial value:
```cpp
= { Vchk_3(discrete_uncertain_set_string,DiscreteUncSetStr) }
```

`Var_check var_mp_check_deur[] [static]`
Initial value:
```cpp
= { Vchk_3(discrete_uncertain_set_real,DiscreteUncSetReal) }
```

`Var_rcheck var_mp_cbound[] [static]`
Initial value:
```cpp
= {
  Vchk_7(continuous_design,ContinuousDes,continuousDesign),
  Vchk_7(continuous_state,ContinuousState,continuousState),
  Vchk_5(normal_uncertain,NormalUnc,normalUnc),
  Vchk_5(lognormal_uncertain,LognormalUnc,lognormalUnc),
  Vchk_5(uniform_uncertain,UniformUnc,uniformUnc),
  Vchk_5(triangular_uncertain,TriangularUnc,triangularUnc),
  Vchk_5(beta_uncertain,BetaUnc,betaUnc) }
```

This is used within check_variables_node(): `Var_RealBoundIPCheck()` is applied to validate bounds and initial points.
Referenced by NIDRProblemDescDB::check_variables_node().

`Var_ichck var_mp_drangef[] [static]`
Initial value:
```cpp
= {
  Vchk_7(discrete_design_range,DiscreteDesRange,discreteDesignRange),
  Vchk_7(discrete_state_range,DiscreteStateRange,discreteStateRange) }
```

This is used in check_variables_node(): `Var_IntBoundIPCheck()` is applied to validate bounds and initial points, and in make_variable_defaults(): `Vgen_*` is called to infer bounds.
Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().

`const char* SCI_FIELD_NAMES[]`
Initial value:
```cpp
```

fields to pass to Scilab in Dakota structure
Referenced by ScilabInterface::scilab_engine_run().
12.2 SIM Namespace Reference

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA.

Classes

- class ParallelDirectApplicInterface
  
  *Sample derived interface class for testing parallel simulator plug-ins using assign_rep().*

- class SerialDirectApplicInterface
  
  *Sample derived interface class for testing serial simulator plug-ins using assign_rep().*

12.2.1 Detailed Description

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA. A typical use of plug-ins with assign_rep() is to publish a simulation interface for use in library mode. See Interfacing with Dakota as a Library for more information.
Chapter 13

Class Documentation

13.1 ActiveSet Class Reference

Container class for active set tracking information. Contains the active set request vector and the derivative variables vector.

Public Member Functions

- **ActiveSet ()**
  *default constructor*

- **ActiveSet (size_t num_fns, size_t num_deriv_vars)**
  *standard constructor*

- **ActiveSet (size_t num_fns)**
  *partial constructor*

- **ActiveSet (const ShortArray &asv, const SizetArray &dvv)**
  *alt constructor*

- **ActiveSet (const ActiveSet &set)**
  *copy constructor*

- **~ActiveSet ()**
  *destructor*

- **ActiveSet & operator= (const ActiveSet &set)**
  *assignment operator*

- **void reshape (size_t num_fns, size_t num_deriv_vars)**
  *reshape requestVector and derivVarsVector*

- **void reshape (size_t num_fns)**
  *reshape requestVector*

- **const ShortArray & request_vector () const**
  *return the request vector*

- **void request_vector (const ShortArray &rv)**
  *set the request vector*

- **void request_values (const short rv_val)**
  *set all request vector values*
• short request_value (const size_t index) const
  get the value of an entry in the request vector
• void request_value (const short rv_val, const size_t index)
  set the value of an entry in the request vector
• const SizetArray & derivative_vector () const
  return the derivative variables vector
• void derivative_vector (const SizetArray &dvv)
  set the derivative variables vector from a SizetArray
• void derivative_vector (SizetMultiArrayConstView dvv)
  set the derivative variables vector from a SizetMultiArrayConstView
• void derivative_start_value (size_t dvv_start_val)
  set the derivative variables vector values
• void read (std::istream &s)
  read an active set object from an std::istream
• void write (std::ostream &s) const
  write an active set object to an std::ostream
• void write_annotated (std::ostream &s) const
  write an active set object to an std::ostream in annotated format
• void read (MPIUnpackBuffer &s)
  read an active set object from a packed MPI buffer
• void write (MPIPackBuffer &s) const
  write an active set object to a packed MPI buffer

Private Member Functions

• template<class Archive >
  void serialize (Archive &ar, const unsigned int version)
  implementation of Boost serialize for ActiveSet

Private Attributes

• ShortArray requestVector
  the vector of response requests
• SizetArray derivVarsVector
  the vector of variable ids used for computing derivatives

Friends

• class boost::serialization::access
• bool operator==(const ActiveSet &set1, const ActiveSet &set2)
  equality operator
• bool operator!=(const ActiveSet &set1, const ActiveSet &set2)
  inequality operator
### 13.1.1 Detailed Description

Container class for active set tracking information. Contains the active set request vector and the derivative variables vector.

The `ActiveSet` class is a small class whose initial design function is to avoid having to pass the ASV and D-VV separately. It is not part of a class hierarchy and does not employ reference-counting/representation-sharing idioms (e.g., handle-body).

### 13.1.2 Member Data Documentation

**ShortArray requestVector** [private]

The vector of response requests

It uses a 0 value for inactive functions and sums 1 (value), 2 (gradient), and 4 (Hessian) for active functions.

Referenced by `ActiveSet::ActiveSet()`, `ActiveSet::operator=()`, `ActiveSet::read()`, `ActiveSet::request_value()`, `ActiveSet::request_values()`, `ActiveSet::request_vector()`, `ActiveSet::reshape()`, `ActiveSet::write()`, and `ActiveSet::write_annotated()`.

**SizetArray derivVarsVector** [private]

The vector of variable ids used for computing derivatives

These ids will generally identify either the active continuous variables or the inactive continuous variables.

Referenced by `ActiveSet::ActiveSet()`, `ActiveSet::derivative_start_value()`, `ActiveSet::derivative_vector()`, `ActiveSet::operator=()`, `ActiveSet::read()`, `ActiveSet::reshape()`, `ActiveSet::write()`, and `ActiveSet::write_annotated()`.

The documentation for this class was generated from the following file:

- DakotaActiveSet.hpp

### 13.2 ActiveSubspaceModel Class Reference

Active subspace model for input (variable space) reduction.

Inheritance diagram for `ActiveSubspaceModel`:

```
Model

RecastModel

ActiveSubspaceModel
```

**Public Member Functions**

- `ActiveSubspaceModel (ProblemDescDB &problem_db)`  
  *Problem database constructor.*
- `ActiveSubspaceModel (const Model &sub_model, unsigned int dimension, const RealMatrix &rotation_matrix, short output_level)`  
  *Lightweight constructor*
- `~ActiveSubspaceModel ()`
CHAPTER 13. CLASS DOCUMENTATION

**Destructor**

- bool initialize_mapping (ParLevLIter pl_iter)
- bool finalize_mapping ()
  finalize model mapping, returns true if the variables size has changed
- bool mapping_initialized () const
  return true if mapping has been fully initialized, false otherwise.
- bool resize_pending () const
  return true if a potential resize is still pending, such that sizing-based initialization should be deferred
- void stop_init_mapping (ParLevLIter pl_iter)
  called from IteratorScheduler::init_iterator() for iteratorComm rank 0 to terminate serve_init_mapping() on other iteratorComm processors
- int serve_init_mapping (ParLevLIter pl_iter)
  called from IteratorScheduler::init_iterator() for iteratorComm rank != 0 to balance resize() calls on iteratorComm rank 0

**Protected Member Functions**

- void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)
- void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)
  portion of set_communicators() specific to derived model classes
- void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)
  portion of free_communicators() specific to derived model classes
- void derived Evaluate (const ActiveSet &set)
  portion of evaluate() specific to derived model classes
- void derived Evaluate_nowait (const ActiveSet &set)
  portion of evaluate_nowait() specific to derived model classes
- const IntResponseMap & derived synchronize ()
  portion of synchronize() specific to derived model classes
- const IntResponseMap & derived synchronize_nowait ()
  portion of synchronize_nowait() specific to derived model classes
- void component_parallel_mode (short mode)
  update component parallel mode for supporting parallelism in the offline and online phases
- void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)
  Service the offline and online phase job requests received from the master; completes when termination message received from stop_servers().
- void stop_servers ()
  Executed by the master to terminate the offline and online phase server operations when iteration on the Active-SubspaceModel is complete.
- Model get_sub_model (ProblemDescDB &problem_db)
  retrieve the sub-Model from the DB to pass up the constructor chain
- void init_fullspace_sampler (unsigned short sample_type)
  initialize the native problem space Monte Carlo sampler
- void validate_inputs ()
  validate the build controls and set defaults
void build_subspace ()
    sample the model’s gradient, computed the SVD, and form the active subspace rotation matrix.

void generate_fullspace_samples (unsigned int diff_samples)
    sample the derivative at diff_samples points and leave temporary in dace_iterator

void populate_matrices (unsigned int diff_samples)
    populate the derivative and vars matrices with fullSpaceSampler samples

void compute_svd ()
    factor the derivative matrix and analyze singular values, assessing convergence and rank, returning whether tolerance met

void identify_subspace ()
    use the truncation methods to identify the size of an active subspace

unsigned int computeBingLiCriterion (RealVector &singular_values)
    compute Bing Li’s criterion to identify the active subspace

unsigned int computeConstantineMetric (RealVector &singular_values)
    compute Constantine’s metric to identify the active subspace

unsigned int computeEnergyCriterion (RealVector &singular_values)
    Compute active subspace size based on eigenvalue energy. Compatible with other truncation methods.

unsigned int computeCrossValidationMetric ()
    Use cross validation of a moving least squares surrogate to identify the size of an active subspace that meets an error tolerance.

Real build_cv_surrogate (Model &cv_surr_model, RealMatrix training_x, IntResponseMap training_y, RealMatrix test_x, IntResponseMap test_y)
    Build moving least squares surrogate over candidate active subspace.

unsigned int determine_rank_cv (const std::vector< Real > &cv_error)

unsigned int min_index (const std::vector< Real > &cv_error)

unsigned int tolerance_met_index (const std::vector< Real > &cv_error, Real tolerance, bool &tol_met)

std::vector< Real > negative_diff (const std::vector< Real > &cv_error)

void build_surrogate ()
    Build surrogate over active subspace.

void initialize_recast ()
    Initialize the base class RecastModel with reduced space variable sizes.

SizetArray variables_resize ()
    Create a variables components totals array with the reduced space size for continuous variables.

void uncertain_vars_to_subspace ()
    translate the characterization of uncertain variables in the native model to the reduced space of the transformed model

void update_linear_constraints ()
    transform the original bounded domain (and any existing linear constraints) into linear constraints in the reduced space

void update_var_labels ()
    update variable labels
Static Protected Member Functions

- static void vars_mapping (const Variables &recast_xi_vars, Variables &sub_model_x_vars)
  map the active continuous recast variables to the active submodel variables (linear transformation)
- static void set_mapping (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  map the inbound ActiveSet to the sub-model (map derivative variables)
- static void response_mapping (const Variables &recast_y_vars, const Variables &sub_model_x_vars, const Response &sub_model_resp, Response &recast_resp)
  map responses from the sub-model to the recast model

Protected Attributes

- int randomSeed
  seed controlling all samplers
- int initialSamples
  initial number of samples at which to query the truth model
- int maxFunctionEvals
  maximum number of build evaluations
- bool subspaceIdBingLi
  Boolean flag signaling use of Bing Li criterion to identify active subspace dimension.
- bool subspaceIdConstantine
  Boolean flag signaling use of Constantine criterion to identify active subspace dimension.
- bool subspaceIdEnergy
  Boolean flag signaling use of eigenvalue energy criterion to identify active subspace dimension.
- bool subspaceIdCV
  Boolean flag signaling use of cross validation to identify active subspace dimension.
- size_t numReplicates
  Number of bootstrap samples for subspace identification.
- bool transformVars
  boolean flag to determine if variables should be transformed to u-space before active subspace initialization
- size_t numFullspaceVars
  Number of fullspace active continuous variables.
- size_t numFunctions
  Total number of response functions.
- unsigned int totalSamples
  total construction samples evaluated so far
- bool subspaceInitialized
  boolean flag to determine if mapping has been fully initialized
- unsigned short subspaceNormalization
  Normalization to use in the case of multiple QoI’s.
- unsigned int reducedRank
  current approximation of system rank
- RealMatrix activeBasis
13.2. ACTIVESUBSPACEMODEL CLASS REFERENCE

- **RealMatrix inactiveBasis**
  basis for the inactive subspace

- **RealVector inactiveVars**
  current inactive variables

- **RealMatrix derivativeMatrix**
  matrix of derivative data with numFunctions columns per fullspace sample; each column contains the gradient of one function at one sample point, so total matrix size is numContinuousVars \( \times \) (numFunctions \( \times \) numSamples)

- **RealMatrix leftSingularVectors**
  matrix of the left singular vectors of derivativeMatrix

- **RealVector singularValues**
  singular values of derivativeMatrix

- **RealMatrix varsMatrix**
  matrix of fullspace variable points samples size numContinuousVars \( \times \) (numSamples)

- **RealArray gradientScaleFactors**
  Gradient scaling factors to make multiple response function gradients similar orders of magnitude.

- **Real truncationTolerance**
  Truncation tolerance for eigenvalue energy subspace identification.

- **bool cvIncremental**

- **short cvIdMethod**

- **Real cvRefTolerance**

- **Real cvDecreaseTolerance**

- **unsigned int cvMaxRank**
  maximum subspace size to consider using cross validation

- **Model surrogateModel**
  model containing a surrogate built over the active subspace

- **bool buildSurrogate**
  flag specifying whether or not a surrogate is built over the subspace

- **int refinementSamples**
  Number of refinement samples to use when building a surrogate.

- **Iterator fullspaceSampler**
  Monte Carlo sampler for the full parameter space.

- **size_t miPLIndex**
  the index of the active metaiterator-iterator parallelism level (corresponding to ParallelConfiguration::miPLIters) used at runtime

- **int onlineEvalConcurrency**
  Concurrency to use once subspace has been built.

- **int offlineEvalConcurrency**
  Concurrency to use when building subspace.

- **IntResponseMap surrResponseMap**
  map of responses returned in buildSurrogate mode

- **IntIntMap surrIdMap**
  map from surrogateModel evaluation ids to RecastModel ids
Static Protected Attributes

- static ActiveSubspaceModel * asmInstance
  
  static pointer to this class for use in static callbacks

Additional Inherited Members

13.2.1 Detailed Description

Active subspace model for input (variable space) reduction.

- Specialization of a RecastModel that identifies an active subspace during build phase and creates a RecastModel in the reduced space

13.2.2 Constructor & Destructor Documentation

ActiveSubspaceModel( const Model & sub_model, unsigned int dimension, const RealMatrix & rotation_matrix, short output_level )

lightweight constructor

An ActiveSubspaceModel will be built over all functions, without differentiating primary vs. secondary constraints. However the associated RecastModel has to differentiate. Currently identifies subspace for continuous variables only, but carries other active variables along for the ride.

References ActiveSubspaceModel::activeBasis, ActiveSubspaceModel::build_surrogate(), ActiveSubspaceModel::buildSurrogate, Model::componentParallelMode, Model::derivative_concurrency(), ActiveSubspaceModel::inactiveBasis, ActiveSubspaceModel::initialize_recast(), ActiveSubspaceModel::initialSamples, Model::modelType, ActiveSubspaceModel::numFullspaceVars, ActiveSubspaceModel::offlineEvalConcurrency, ActiveSubspaceModel::onlineEvalConcurrency, Model::output_level(), Model::outputLevel, ActiveSubspaceModel::reducedRank, RecastModel::subModel, ActiveSubspaceModel::subspaceInitialized, Model::supportsEstimDerivs, ActiveSubspaceModel::uncertain_vars_to_subspace(), ActiveSubspaceModel::update_linear_constraints(), and ActiveSubspaceModel::update_var_labels().

13.2.3 Member Function Documentation

bool initialize_mapping( ParLevLIter pl_iter ) [virtual]

May eventually take on init_comms and related operations. Also may want ide of build/update like DataFitSurrModel, eventually.

Reimplemented from Model.

References ActiveSubspaceModel::build_subspace(), ActiveSubspaceModel::build_surrogate(), ActiveSubspaceModel::buildSurrogate, ActiveSubspaceModel::component_parallel_mode(), RecastModel::initialize_mapping(), Model::initialize_mapping(), ActiveSubspaceModel::initialize_recast(), ActiveSubspaceModel::miPLIndex, Model::modelPCIter, ActiveSubspaceModel::numFullspaceVars, Model::outputLevel, ActiveSubspaceModel::reducedRank, RecastModel::subModel, ActiveSubspaceModel::subspaceInitialized, Model::supportsEstimDerivs, ActiveSubspaceModel::uncertain_vars_to_subspace(), ActiveSubspaceModel::update_linear_constraints(), and ActiveSubspaceModel::update_var_labels().

void derived_init_communicators( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag ) [protected], [virtual]

This specialization is because the model is used in multiple contexts in this iterator, depending on build phase. Note that this overrides the default behavior at Iterator which recurses into any submodels.
13.2. ACTIVESUBSPACEMODEL CLASS REFERENCE

Reimplemented from Model.
References ActiveSubspaceModel::fullspaceSampler, Iterator::init_communicators(), Model::init_communicators(),
ActiveSubspaceModel::onlineEvalConcurrency, RecastModel::subModel, and ActiveSubspaceModel::subspace-Initialized.

Real build_cv_surrogate ( Model & cv_surr_model, RealMatrix training_x, IntResponseMap training_y,
RealMatrix test_x, IntResponseMap test_y ) [protected]

Build moving least squares surrogate over candidate active subspace.

Build global moving least squares surrogate model to use in cross validation to estimate active subspace size.
References Response::active_set(), Model::continuous_variables(), Response::copy(), Model::current_response(),
Model::evaluate(), ActiveSubspaceModel::numFunctions, and Model::update_approximation().
Referenced by ActiveSubspaceModel::computeCrossValidationMetric().

void build_surrogate ( ) [protected]

Build surrogate over active subspace.

Build surrogate over active subspace
References Response::active_set(), Iterator::active_set(), ActiveSubspaceModel::activeBasis, Iterator::all_responses(),
Iterator::all_samples(), Model::append_approximation(), ActiveSubspaceModel::asmInstance, Model::assign_rep(),
Model::current_response(), ActiveSubspaceModel::fullspaceSampler, ActiveSubspaceModel::miPLIndex, Model-
::modelPCIter, Model::outputLevel, ActiveSubspaceModel::reducedRank, ActiveSubspaceModel::refinementSamples,
ActiveSet::request_values(), Iterator::run(), Iterator::sampling_reference(), Iterator::sampling_reset(), and Active-
SubspaceModel::surrogateModel.
Referenced by ActiveSubspaceModel::ActiveSubspaceModel(), and ActiveSubspaceModel::initialize_mapping().

void initialize_recast ( ) [protected]

Initialize the base class RecastModel with reduced space variable sizes.

Initialize the recast model based on the reduced space, with no response function mapping (for now). TODO:
use a surrogate model over the inactive dimension.
References Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Response::function-
gradients(), Response::function_hessians(), RecastModel::init_maps(), RecastModel::init_sizes(), Model::num-
functions(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), ActiveSubspaceModel::num-
Functions, ActiveSubspaceModel::reducedRank, ActiveSubspaceModel::response_mapping(), ActiveSubspace-
Model::set_mapping(), RecastModel::subModel, ActiveSubspaceModel::variables_resize(), and ActiveSubspace-
Model::vars_mapping().
Referenced by ActiveSubspaceModel::ActiveSubspaceModel(), and ActiveSubspaceModel::initialize_mapping().

void uncertain_vars_to_subspace ( ) [protected]

translate the characterization of uncertain variables in the native_model to the reduced space of the transformed model

transform and set the distribution parameters in the reduced model
Convert the user-specified normal random variables to the appropriate reduced space variables, based on the
orthogonal transformation.
TODO: Generalize to convert other random variable types (non-normal)
TODO: The translation of the correlations from full to reduced space is likely wrong for rank correlations;
should be correct for covariance.
References ActiveSubspaceModel::ActiveSubspaceModel(), and ActiveSubspaceModel::initialize_mapping().
void vars_mapping ( const Variables & recast_y_vars, Variables & sub_model_x_vars ) [static], [protected]
map the active continuous recast variables to the active submodel variables (linear transformation).
Perform the variables mapping from recast reduced dimension variables \( y \) to original model \( x \) variables via linear transformation. Maps only continuous variables.
References ActiveSubspaceModel::activeBasis, ActiveSubspaceModel::asmInstance, Variables::continuous_variables(), Dakota::copy_data(), ActiveSubspaceModel::inactiveBasis, ActiveSubspaceModel::inactiveVars, and Model::outputLevel.
Referenced by ActiveSubspaceModel::initialize_recast().

void set_mapping ( const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set ) [static], [protected]
map the inbound ActiveSet to the sub-model (map derivative variables)
Simplified derivative variables mapping where all continuous depend on all others. TODO: Could instead rely on a richer default in RecastModel based on varsMapIndices.
References ActiveSubspaceModel::asmInstance, Variables::cv(), Model::cv(), ActiveSet::derivative_vector(), and RecastModel::subModel.
Referenced by ActiveSubspaceModel::initialize_recast().

void response_mapping ( const Variables & recast_y_vars, const Variables & sub_model_x_vars, const Response & sub_model_resp, Response & recast_resp ) [static], [protected]
map responses from the sub-model to the recast model
Perform the response mapping from submodel to recast response
References ActiveSubspaceModel::activeBasis, ActiveSubspaceModel::asmInstance, Response::function_gradients(), Response::function_hessians(), and Response::function_values().
Referenced by ActiveSubspaceModel::initialize_recast().

13.2.4 Member Data Documentation

ActiveSubspaceModel * asmInstance [static], [protected]
static pointer to this class for use in static callbacks
initialization of static needed by RecastModel
Referenced by ActiveSubspaceModel::ActiveSubspaceModel(), ActiveSubspaceModel::build_surrogate(), ActiveSubspaceModel::response_mapping(), ActiveSubspaceModel::set_mapping(), and ActiveSubspaceModel::vars_mapping().
The documentation for this class was generated from the following files:
- ActiveSubspaceModel.hpp
- ActiveSubspaceModel.cpp

13.3 AdaptedBasisModel Class Reference

Adapted basis model for input (variable space) reduction.
Inheritance diagram for AdaptedBasisModel:
Public Member Functions

- **AdaptedBasisModel (ProblemDescDB &problem_db)**
  Problem database constructor.
- **~AdaptedBasisModel ()**
  Destructor
- **bool initialize_mapping (ParLevLIter pl_iter)**
  Initialize model mapping, returns true if the variables size has changed
- **bool finalize_mapping ()**
  Finalize model mapping, returns true if the variables size has changed
- **bool mapping_initialized () const**
  Return true if mapping has been fully initialized, false otherwise.
- **bool resize_pending () const**
  Return true if a potential resize is still pending, such that sizing-based initialization should be deferred
- **void stop_init_mapping (ParLevLIter pl_iter)**
  Called from IteratorScheduler::init_iterator() for iteratorComm rank 0 to terminate serve_init_mapping() on other iteratorComm processors
- **int serve_init_mapping (ParLevLIter pl_iter)**
  Called from IteratorScheduler::init_iterator() for iteratorComm rank != 0 to balance resize() calls on iteratorComm rank 0

Protected Member Functions

- **void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)**
  Portion of set_communicators() specific to derived model classes
- **void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)**
  Portion of set_communicators() specific to derived model classes
- **void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)**
  Portion of free_communicators() specific to derived model classes
- **void derived_evaluate (const ActiveSet &set)**
  Portion of evaluate() specific to derived model classes
- **void derived_evaluate_nowait (const ActiveSet &set)**
  Portion of evaluate_nowait() specific to derived model classes
- **const IntResponseMap & derived_synchronize ()**
  Portion of synchronize() specific to derived model classes
- **const IntResponseMap & derived_synchronize_nowait ()**
  Portion of synchronize_nowait() specific to derived model classes
• **void component_parallel_mode** (short mode)
  
  Update component parallel mode for supporting parallelism in the offline and online phases

• **void serve_run** (ParLevLIter pl_iter, int max_eval_concurrency)
  
  Service the offline and online phase job requests received from the master; completes when termination message received from **stop_servers**().

• **void stop_servers** ()
  
  Executed by the master to terminate the offline and online phase server operations when iteration on the **Adapted-BasisModel** is complete.

• **Model get_sub_model** (ProblemDescDB &problem_db)
  
  Retrieve the sub-Model from the DB to pass up the constructor chain

• **void validate_inputs** ()
  
  Validate the build controls and set defaults

• **void identify_subspace** ()

• **void initialize_recast** ()
  
  Initialize the base class **RecastModel** with reduced space variable sizes.

• **SizetArray variables_resize** ()
  
  Create a variables components totals array with the reduced space size for continuous variables.

• **void uncertain-vars-to-subspace** ()
  
  Translate the characterization of uncertain variables in the native model to the reduced space of the transformed model

### Static Protected Member Functions

• static **void vars_mapping** (const Variables &recast_xi_vars, Variables &sub_model_x_vars)
  
  Map the active continuous recast variables to the active submodel variables (linear transformation)

• static **void set_mapping** (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  
  Map the inbound **ActiveSet** to the sub-model (map derivative variables)

• static **void response_mapping** (const Variables &recast_y_vars, const Variables &sub_model_x_vars, const Response &sub_model_resp, Response &recast_resp)
  
  Map responses from the sub-model to the recast model

### Protected Attributes

• **Iterator * pcePilotExpRepPtr**
  
  PCE representation pointer that is initialized in **get_sub_model**() and then assigned into pcePilotExpansion in the constructor initializer list.

• **Iterator pcePilotExpansion**
  
  Low-order (linear or quadratic) PCE generator for computing rotation matrices $A_j$ for each of the QoI; this is low-order and potentially high-dimension whereas a client PCE could be high-order in the reduced dimension

• **size_t numFullspaceVars**
  
  Number of fullspace active continuous variables.

• **size_t numFunctions**
  
  Total number of response functions.

• **bool adaptedBasisInitialized**
13.3. ADAPTEDBASISMODEL CLASS REFERENCE

**boolean flag to determine if mapping has been fully initialized**

- **unsigned int reducedRank**
  
  current approximation of system rank

- **RealMatrix rotationMatrix**
  
  basis for the reduced subspace

- **size_t miPLIndex**
  
  the index of the active metaiterator-iterator parallelism level (corresponding to ParallelConfiguration::miPLIters) used at runtime

- **int onlineEvalConcurrency**
  
  Concurrency to use once subspace has been built.

- **int offlineEvalConcurrency**
  
  Concurrency to use when building subspace.

**Static Protected Attributes**

- **static AdaptedBasisModel * abmInstance**

  static pointer to this class for use in static callbacks

**Additional Inherited Members**

**13.3.1 Detailed Description**

Adapted basis model for input (variable space) reduction.

Specialization of a RecastModel that creates an adapted basis model during build phase and creates a RecastModel in the reduced space

**13.3.2 Member Function Documentation**

```cpp
void derived_init_communicators ( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag )
[inline], [protected], [virtual]
```

This specialization is because the model is used in multiple contexts depending on build phase.

Reimplemented from Model.

References Iterator::init_communicators(), Model::init_communicators(), AdaptedBasisModel::onlineEvalConcurrency, AdaptedBasisModel::pcePilotExpansion, and RecastModel::subModel.

```cpp
void initialize_recast ( ) [protected]
```

Initialize the base class RecastModel with reduced space variable sizes.

Initialize the recast model based on the reduced space, with no response function mapping (for now).

References Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Response::function_gradients(), Response::function_hessians(), RecastModel::init_maps(), RecastModel::init_sizes(), Model::num_functions(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), AdaptedBasisModel::reduced_Rank, AdaptedBasisModel::response_mapping(), AdaptedBasisModel::set_mapping(), RecastModel::subModel, AdaptedBasisModel::variables_resize(), and AdaptedBasisModel::vars_mapping().

Referenced by AdaptedBasisModel::initialize_mapping().
void uncertain_vars_to_subspace( ) [protected]
translate the characterization of uncertain variables in the native_model to the reduced space of the transformed model
- transform and set the distribution parameters in the reduced model
- Convert the user-specified normal random variables to the appropriate reduced space variables, based on the orthogonal transformation.
- TODO: Generalize to convert other random variable types (non-normal)
- TODO: The translation of the correlations from full to reduced space is likely wrong for rank correlations; should be correct for covariance.
  Referenced by AdaptedBasisModel::initialize_mapping().

void vars_mapping ( const Variables & recast_y_vars, Variables & sub_model_x_vars ) [static], [protected]
map the active continuous recast variables to the active submodel variables (linear transformation)
- Perform the variables mapping from recast reduced dimension variables y to original model x variables via linear transformation. Maps only continuous variables.
  References AdaptedBasisModel::abmInstance, Variables::continuous_variables(), Variables::continuous_variables_view(), Model::outputLevel, and AdaptedBasisModel::rotationMatrix.
  Referenced by AdaptedBasisModel::initialize_recast().

void set_mapping ( const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set ) [static], [protected]
map the inbound ActiveSet to the sub-model (map derivative variables)
- Simplified derivative variables mapping where all continuous depend on all others. TODO: Could instead rely on a richer default in RecastModel based on varsMapIndices.
  References AdaptedBasisModel::abmInstance, Variables::cv(), Model::cv(), ActiveSet::derivative_vector(), and RecastModel::subModel.
  Referenced by AdaptedBasisModel::initialize_recast().

void response_mapping ( const Variables & recast_y_vars, const Variables & sub_model_x_vars, const Response & sub_model_resp, Response & recast_resp ) [static], [protected]
map responses from the sub-model to the recast model
- Perform the response mapping from submodel to recast response
  References AdaptedBasisModel::abmInstance, Response::function_gradients(), Response::function_hessians(), Response::function_values(), and AdaptedBasisModel::rotationMatrix.
  Referenced by AdaptedBasisModel::initialize_recast().

13.3.3 Member Data Documentation

AdaptedBasisModel * abmInstance [static], [protected]
static pointer to this class for use in static callbacks
- initialization of static needed by RecastModel
  Referenced by AdaptedBasisModel::AdaptedBasisModel(), AdaptedBasisModel::response_mapping(), AdaptedBasisModel::set_mapping(), and AdaptedBasisModel::vars_mapping().
  The documentation for this class was generated from the following files:
13.4 Analyzer Class Reference

Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy.

Inheritance diagram for Analyzer:

```
Analyzer
   |__Iterator
   |__NonD
      |__NonDCalibration
      |__NonDExpansion
      |__NonDIntegration
      |__NonDInterval
      |__NonDPOFDarts
      |__NonDReliability
      |__NonDRKDDarts
      |__NonDSampling
   |__ParamStudy
      |__PSUADEDesignCompExp
      |__RichExtrapVerification
   |__FSUDesignCompExp
   |__DDACEDesignCompExp
```

**Public Member Functions**

- `const VariablesArray & all_variables ()`
  
  return the complete set of evaluated variables

- `const RealMatrix & all_samples ()`
  
  return the complete set of evaluated samples

- `const IntResponseMap & all_responses () const`
  
  return the complete set of computed responses

- `bool resize ()`
  
  reinitializes iterator based on new variable size

- `int num_samples () const`

- `virtual void vary_pattern (bool pattern_flag)`
  
  sets varyPattern in derived classes that support it

**Protected Member Functions**

- `Analyzer ()`
  
  default constructor

- `Analyzer (ProblemDescDB &problem_db, Model &model)`
CHAPTER 13. CLASS DOCUMENTATION

- **Analyzer** (unsigned short method_name, Model &model)
  - alternate constructor for instantiations “on the fly” with a Model
- **Analyzer** (unsigned short method_name)
  - alternate constructor for instantiations “on the fly” without a Model
- **∼Analyzer** ()
  - destructor
- virtual void get_parameter_sets (Model &model)
  - Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
- virtual void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix)
  - Generate one block of numSamples samples (ndim * num_samples), populating design_matrix.
- virtual void update_model_from_sample (Model &model, const Real *sample_vars)
  - update model’s current variables with data from sample
- virtual void update_model_from_variables (Model &model, const Variables &vars)
  - update model’s current variables with data from vars
- virtual void sample_to_variables (const Real *sample_vars, Variables &vars)
  - convert column of samples array to variables; derived classes may reimplement for more than active continuous variables
- void update_from_model (const Model &model)
  - set inherited data attributes based on extractions from incoming model
- void initialize_run ()
  - utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
- void pre_run ()
  - pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void post_run (std::ostream &s)
  - post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- void finalize_run ()
  - utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- void pre_output ()
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  - print the final iterator results
- const Model & algorithm_space_model () const
- const Variables & variables_results () const
  - return a single final iterator solution (variables)
- const Response & response_results () const
  - return a single final iterator solution (response)
- const VariablesArray & variables_array_results ()
  - return multiple final iterator solutions (variables). This should only be used if returns_multiple_points() returns true.
- const ResponseArray & response_array_results ()
13.4. ANALYZER CLASS REFERENCE

return multiple final iterator solutions (response). This should only be used if returns_multiple_points() returns true.

- **void response_results_active_set (const ActiveSet &set)**
  set the requested data for the final iterator response results

- **bool compact_mode () const**
  returns Analyzer::compactMode

- **bool returns_multiple_points () const**
  indicates if this iterator returns multiple final points. Default return is false. Override to return true if appropriate.

- **void evaluate_parameter_sets (Model &model, bool log resp_flag, bool log_best_flag)**
  perform function evaluations to map parameter sets (allVariables) into response sets (allResponses)

- **void get_vbd_parameter_sets (Model &model, int num_samples)**
  generate replicate parameter sets for use in variance-based decomposition

- **void compute_vbd_stats (const int num_samples, const IntResponseMap &resp_samples)**
  compute VBD-based Sobol indices

- **void read_variables_responses (int num_evals, size_t num_vars)**
  convenience function for reading variables/responses (used in derived classes post_input)

- **void print_sobol_indices (std::ostream &s) const**
  Printing of VBD results.

- **void samples_to_variables_array (const RealMatrix &sample_matrix, VariablesArray &vars_array)**
  convert samples array to variables array; e.g., allSamples to allVariables

- **virtual void variables_to_sample (const Variables &vars, Real *sample_c vars)**
  convert the active continuous variables into a column of allSamples

- **void variables_array_to_samples (const VariablesArray &vars_array, RealMatrix &sample_matrix)**
  convert variables array to samples array; e.g., allVariables to allSamples

**Protected Attributes**

- **size_t numFunctions**
  number of response functions

- **size_t numContinuousVars**
  number of active continuous vars

- **size_t numDiscreteIntVars**
  number of active discrete integer vars

- **size_t numDiscreteStringVars**
  number of active discrete string vars

- **size_t numDiscreteRealVars**
  number of active discrete real vars

- **bool compactMode**
  switch for allSamples (compact mode) instead of allVariables (normal mode)

- **VariablesArray allVariables**
  array of all variables to be evaluated in evaluate_parameter_sets()

- **RealMatrix allSamples**
  compact alternative to allVariables

- **IntResponseMap allResponses**
array of all responses to be computed in `evaluate_parameter_sets`

- **StringArray** `allHeaders`
  array of headers to insert into output while evaluating `allVariables`

- **size_t** `numObjFns`
  number of objective functions

- **size_t** `numLsqTerms`
  number of least squares terms

- **RealPairPRPMultiMap** `bestVarsRespMap`
  map which stores best set of solutions

### Private Member Functions

- **void** `compute_best_metrics` (const `Response` &response, std::pair<Real, Real> &metrics)
  compares current evaluation to best evaluation and updates best

- **void** `update_best` (const `Variables` &vars, int eval_id, const `Response` &response)
  compares current evaluation to best evaluation and updates best

- **void** `update_best` (const `Real` *sample_c_vars, int eval_id, const `Response` &response)
  compares current evaluation to best evaluation and updates best

### Private Attributes

- **int** `writePrecision`
  write precision as specified by the user

- **Real** `vbdDropTol`
  tolerance for omitting output of small VBD indices

- **RealVectorArray** `S4`
  VBD main effect indices.

- **RealVectorArray** `T4`
  VBD total effect indices.

### Additional Inherited Members

#### 13.4.1 Detailed Description

Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy.

The Analyzer class provides common data and functionality for various types of systems analysis, including nondeterministic analysis, design of experiments, and parameter studies.

#### 13.4.2 Member Function Documentation

**int num_samples () const [inline], [virtual]**

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from `Iterator`.

Reimplemented in NonDSampling, NonDQuadrature, NonDSparseGrid, NonDCubature, DDACEDesign-CompExp, FSUDesignCompExp, and PSUADEDesignCompExp.
13.4. ANALYZER CLASS REFERENCE

References Model::derivative_concurrency(), Iterator::iteratedModel, and Iterator::maxEvalConcurrency.
Referenced by NonDDREAMBayesCalibration::archive_acceptance_chain(), NonDDREAMBayesCalibration::cache_chain(), NonDBayesCalibration::compute_statistics(), Analyzer::compute_vbd_stats(), NonDBayesCalibration::filter_chain(), NonDGlobalReliability::get_best_sample(), Analyzer::get_vbd_parameter_sets(), NonDPolynomialChaos::ratio_samples_to_order(), Analyzer::samples_to_variables_array(), and Analyzer::variables_array_to_samples().

```cpp
void sample_to_variables ( const Real * sample_c_vars, Variables & vars ) [protected], [virtual]
```

convert column of samples array to variables; derived classes may reimplement for more than active continuous variables
- Default mapping that maps into continuous part of Variables only
  Reimplemented in NonDSampling.
- References Variables::adiv(), Variables::adrv(), Variables::all_discrete_int_variables(), Variables::all_discrete_real_variables(), Variables::continuous_variable(), Model::current_variables(), Variables::inactive_continuous_variables(), Variables::is_null(), Iterator::iteratedModel, Analyzer::numContinuousVars, and Variables::shared_data().
  Referenced by NonDLHSEvidence::post_process_samples(), Analyzer::pre_output(), Analyzer::samples_to_variables_array(), and Analyzer::update_best().

```cpp
void initialize_run ( ) [protected], [virtual]
```

utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
- Perform initialization phases of run sequence prior to pre_run(); commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.
  Reimplemented from Iterator.
  Reimplemented in NonD.
- References Model::initialize_mapping(), Model::is_null(), Iterator::iteratedModel, Model::mapping_initialized(), Iterator::methodPCIter, Analyzer::resize(), Model::set_evaluation_reference(), and Iterator::summaryOutputFlag.
  Referenced by NonD::initialize_run().

```cpp
void pre_run ( ) [protected], [virtual]
```

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
  Reimplemented from Iterator.
  Reimplemented in NonDSampling, NonD, NonDLHSSampling, DDACEDesignCompExp, NonDRKDDarts, FSUDesignCompExp, ParamStudy, NonDMultilevelSampling, and PSUADEDesignCompExp.
- References Analyzer::bestVarsRespMap.
  Referenced by PSUADEDesignCompExp::pre_run(), ParamStudy::pre_run(), FSUDesignCompExp::pre_run(), DDACEDesignCompExp::pre_run(), and NonD::pre_run().

```cpp
void post_run ( std::ostream & s ) [protected], [virtual]
```

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
 CHAPTER 13. CLASS DOCUMENTATION

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.

References Model::is_null(), Iterator::iteratedModel, Model::print_evaluation_summary(), Analyzer::print_results(), Iterator::resultsDB, Iterator::summaryOutputFlag, and ResultsManager::write_databases().

Referenced by NonDMultilevelSampling::post_run(), PSUADEDesignCompExp::post_run(), ParamStudy::post_run(), DDACEDesignCompExp::post_run(), NonDLHSSampling::post_run(), and NonDRKDDarts::post_run().

```cpp
void finalize_run ( ) [protected], [virtual]
```

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub(iterator) executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in NonD.

References Model::finalize_mapping(), Iterator::finalize_run(), Iterator::iteratedModel, Model::mapping_initialized(), and Analyzer::resize().

Referenced by NonD::finalize_run().

```cpp
void pre_output ( ) [protected], [virtual]
```

Generate tabular output with active variables (compactMode) or all variables with their labels and response labels, with no data. Variables are sequenced \{cv, div, drv\}

Reimplemented from Iterator.

References Analyzer::allSamples, Analyzer::allVariables, ParallelLibrary::command_line_pre_run_output(), ParallelLibrary::command_line_user_modes(), Analyzer::compactMode, Variables::copy(), Model::current_response(), Model::current_variables(), Model::interface_id(), Iterator::iteratedModel, Iterator::outputLevel, Iterator::parallelLib, ProgramOptions::pre_run_output_format(), ParallelLibrary::program_options(), Analyzer::sample_to_variables(), Dakota::write_precision, Variables::write_tabular(), and Analyzer::writePrecision.

```cpp
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
```

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Iterator.

Reimplemented in NonDPolynomialChaos, NonDGPMSABayesCalibration, NonDLHSSampling, NonDBayesCalibration, NonDPOFDarts, NonDMultilevelPolynomialChaos, NonDLocalReliability, NonDWASABIBayesCalibration, NonDQUESTOBayesCalibration, NonDAdaptiveSampling, NonDGPImpSampling, NonDAdaptImpSampling, NonDExpansion, NonDInterval, NonDMultilevelSampling, PStudyDACE, Verification, NonDGlobalReliability, and RichExtrapVerification.

References Analyzer::bestVarsRespMap, ParamResponsePair::eval_id(), Response::function_values(), Analyzer::numLSqTerms, Analyzer::numObjFns, ParamResponsePair::response(), and ParamResponsePair::variables().
13.4. ANALYZER CLASS REFERENCE

Referenced by Analyzer::post_run(), Verification::print_results(), PStudyDACE::print_results(), and NonDLHSSampling::print_results().

const Model & algorithm_space_model() const [inline, [protected], [virtual]]
default definition that gets redefined in selected derived Minimizers
Reimplemented from Iterator.
Reimplemented in NonDBayesCalibration, NonDExpansion, NonDGlobalInterval, and NonDReliability.
References Iterator::iteratedModel.

void evaluate_parameter_sets ( Model & model, bool log resp flag, bool log best flag ) [protected]
perform function evaluations to map parameter sets (allVariables) into response sets (allResponses)
Convenience function for derived classes with sets of function evaluations to perform (e.g., NonDSampling, DDACEDesignCompExp, FSUDesignCompExp, ParamStudy).
References Iterator::iteratedModel, Analyzer::allHeaders, Analyzer::allResponses, Analyzer::allVariables, Model::asynch_flag(), Model::compactMode, Response::copy(), Model::current_response(), Model::current_variables(), Model::evaluate(), Model::evaluate_nowait(), Model::evaluation_id(), Model::synchronize(), Analyzer::update_best(), Analyzer::update_model_from_sample(), and Analyzer::update_model_from_variables().
Referenced by ParamStudy::core_run(), PSUADEDesignCompExp::core_run(), FSUDesignCompExp::core_run(), NonDAdeptImpSampling::core_run(), DDAcEDesignCompExp::core_run(), NonDLHSSampling::core_run(), NonDIntegration::core_run(), NonDSampling::core_run(), NonDSparseGrid::evaluate_grid_increment(), NonDSparseGrid::evaluate_set(), NonDMultilevelSampling::lf_increment(), NonDMultilevelSampling::multilevel_control_variate_mc_Qcorr(), NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonDMultilevelSampling::multilevel_mc_Qsum(), NonDMultilevelSampling::multilevel_mc_Ysum(), NonDMultilevelSampling::shared_increment(), and NonDQUESOBayesCalibration::update_model().

void get_vbd_parameter_sets ( Model & model, int num_samples ) [protected]
generate replicate parameter sets for use in variance-based decomposition
Generate (numvars + 2)*num_samples replicate sets for VBD, populating allSamples( numvars, (numvars + 2)*num_samples )
References Dakota::abort_handler(), Analyzer::allSamples, Analyzer::compactMode, Analyzer::get_parameter_sets(), Analyzer::num_samples(), Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, and Analyzer::vary_pattern().
Referenced by FSUDesignCompExp::pre_run(), DDACEDesignCompExp::pre_run(), and NonDLHSSampling::pre_run().

void compute_vbd_stats ( const int num_samples, const IntResponseMap & resp_samples ) [protected]
compute VBD-based Sobol indices
Calculation of sensitivity indices obtained by variance based decomposition. These indices are obtained by the Saltelli version of the Sobol VBD which uses (K+2)*N function evaluations, where K is the number of dimensions (uncertain vars) and N is the number of samples.
References Dakota::abort_handler(), Analyzer::allSamples, Analyzer::num_samples(), Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, Analyzer::numFunctions, Analyzer::S4, and Analyzer::T4.
Referenced by FSUDesignCompExp::post_run(), DDACEDesignCompExp::post_run(), and NonDLHSSampling::post_run().
void read_variables_responses ( int num_evals, size_t num_vars ) [protected]

convenience function for reading variables/responses (used in derived classes post_input)
    read num_evals variables/responses from file

    References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Analyzer::allVariables,
        ParallelLibrary::command_line_post_run_input(), ParallelLibrary::command_line_user_modes(), Analyzer::compactMode, Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Dakota::data_pairs, ParamResponsePair::eval_id(), Model::evaluation_cache(), Iterator::iteratedModel, Model::manage_data_recastings(), Analyzer::numLSqTerms, Analyzer::numObjFns, Iterator::outputLevel, Iterator::parallelLib, ProgramOptions::post_run_input_format(), ParallelLibrary::program_options(), ParamResponsePair::response(), Model::restart_file(), Analyzer::update_best(), Model::user_space_to_iterator_space(), ParamResponsePair::variables(), Analyzer::variables_to_sample(), and ParallelLibrary::write_restart().

    Referenced by PSUADEDesignCompExp::post_input(), ParamStudy::post_input(), FSUDesignCompExp::post_input(), DDACEDesignCompExp::post_input(), and NonDLHSSampling::post_input().

void print_sobol_indices ( std::ostream & s ) const [protected]

Printing of VBD results.
    printing of variance based decomposition indices.

    References Model::continuous_variable_labels(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), Iterator::iteratedModel, Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numFunctions, Model::response_labels(), Analyzer::S4, Analyzer::T4, Analyzer::vbdDropTol, and Dakota::write_precision.

    Referenced by PStudyDACE::print_results(), and NonDLHSSampling::print_results().

void variables_to_sample ( const Variables & vars, Real * sample_c_vars ) [protected], [virtual]

convert the active continuous variables into a column of allSamples
    Default implementation maps active continuous variables only
    Reimplemented in NonDSampling.

    References Variables::continuous_variables(), and Analyzer::numContinuousVars.

    Referenced by Analyzer::read_variables_responses(), and Analyzer::variables_array_to_samples().

The documentation for this class was generated from the following files:

- DakotaAnalyzer.hpp
- DakotaAnalyzer.cpp

13.5 ApplicationInterface Class Reference

Derived class within the interface class hierarchy for supporting interfaces to simulation codes.

Inheritance diagram for ApplicationInterface:
### Public Member Functions

- **ApplicationInterface (const ProblemDescDB &problem_db)**  
  * constructor*
- **∼ApplicationInterface ()**  
  * destructor*

### Protected Member Functions

- **void init_communicators (const IntArray &message_lengths, int max_eval_concurrency)**  
  * allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.*
- **void set_communicators (const IntArray &message_lengths, int max_eval_concurrency)**  
  * set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).*
- **void init_serial ()**
- **int async_local_evaluation_concurrency () const**  
  * return asynchLocalEvalConcurrency*
- **short interface_synchronization () const**  
  * return interfaceSynchronization*
- **bool evaluation_cache () const**  
  * return evalCacheFlag*
- **bool restart_file () const**  
  * return evalCacheFlag*
- **void map (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)**  
  * Provides a "mapping" of variables to responses using a simulation. Protected due to Interface letter-envelope idiom.*
- **void manage_failure (const Variables &vars, const ActiveSet &set, Response &response, int failed_eval_id)**  
  * manages a simulation failure using abort/retry/recover/continuation*
- **const IntResponseMap & synchronize ()**
executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns all jobs
• const IntResponseMap & synchronize_nowait()
  executes a nonblocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns a
  partial set of completed jobs
• void serve_evaluations()
  run on evaluation servers to serve the iterator master
• void stop_evaluation_servers()
  used by the iterator master to terminate evaluation servers
• bool check_multiprocessor_analysis(bool warn)
  checks on multiprocessor analysis configuration
• bool check_asynchronous(bool warn, int max_eval_concurrency)
  checks on asynchronous configuration (for direct interfaces)
• bool check_multiprocessor_asynchronous(bool warn, int max_eval_concurrency)
  checks on asynchronous settings for multiprocessor partitions
• virtual void derived_map(const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an
  evaluation that is specific to a derived class.
• virtual void derived_map_asynch(const ParamResponsePair &pair)
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing
  an asynchronous evaluation that is specific to a derived class.
• virtual void wait_local_evaluations(PRPRQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results.
  It provides the processing code that is specific to derived classes. This version waits for at least one completion.
• virtual void test_local_evaluations(PRPRQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It
  provides the processing code that is specific to derived classes. This version is nonblocking and will return without
  any completions if none are immediately available.
• virtual void init_communicators_checks(int max_eval_concurrency)
  perform construct-time error checks on the parallel configuration
• virtual void set_communicators_checks(int max_eval_concurrency)
  perform run-time error checks on the parallel configuration
• void master_dynamic_schedule_analyses()
  blocking dynamic schedule of all analyses within a function evaluation using message passing
• void serve_analyses_synch()
  serve the master analysis scheduler and manage one synchronous analysis job at a time
• virtual int synchronous_local_analysis(int analysis_id)
  Execute a particular analysis (identified by analysis_id) synchronously on the local processor. Used for the derived
  class specifics within ApplicationInterface::serve_analyses_synch().

Protected Attributes

• ParallelLibrary & parallelLib
  reference to the ParallelLibrary object used to manage MPI partitions for the concurrent evaluations and concur-
  rent analyses parallelism levels
• bool suppressOutput
flag for suppressing output on slave processors

- int `evalCommSize`
  size of `evalComm`
- int `evalCommRank`
  processor rank within `evalComm`
- int `evalServerId`
  evaluation server identifier
- bool `eaDedMasterFlag`
  flag for dedicated master partitioning at ea level
- int `analysisCommSize`
  size of `analysisComm`
- int `analysisCommRank`
  processor rank within `analysisComm`
- int `analysisServerId`
  analysis server identifier
- int `numAnalysisServers`
  current number of analysis servers
- bool `multiProcAnalysisFlag`
  flag for multiprocessor analysis partitions
- bool `asynchLocalAnalysisFlag`
  flag for asynchronous local parallelism of analyses
- int `asynchLocalConcurrency`
  limits the number of concurrent analyses in asynchronous local scheduling and specifies hybrid concurrency when message passing
- int `asynchLocalEvalConcSpec`
  user specification for asynchronous local evaluation concurrency
- int `asynchLocalAnalysisConcSpec`
  user specification for asynchronous local analysis concurrency
- int `numAnalysisDrivers`
  the number of analysis drivers used for each function evaluation (from the analysis_drivers interface specification)
- IntSet `completionSet`
  the set of completed fn_eval_id’s populated by `wait_local_evaluations()` and `test_local_evaluations()`
- String `failureMessage`
  base message for managing failed evals; will be followed with more details in screen output

### Private Member Functions

- bool `duplication_detect` (const `Variables &vars, Response &response, bool asynch_flag)`
  checks data pairs and beforeSynchCorePRPQueue to see if the current evaluation request has already been performed or queued
- void `init_default_asv` (size_t `num_fns`)
  initialize default ASV if needed; this is done at run time due to post-construct time `Response` size changes.
- void `master_dynamic_schedule_evaluations()`
void peer_static_schedule_evaluations ()
blocking static schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

void peer_dynamic_schedule_evaluations ()
blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

void asynchronous_local_evaluations (PRPQueue &prp_queue)
perform all jobs in prp_queue using asynchronous approaches on the local processor

void synchronous_local_evaluations (PRPQueue &prp_queue)
perform all jobs in prp_queue using synchronous approaches on the local processor

void master_dynamic_schedule_evaluations_nowait ()
execute a nonblocking dynamic schedule in a master-slave partition

void peer_static_schedule_evaluations_nowait ()
excecut e a nonblocking static schedule in a peer partition

void peer_dynamic_schedule_evaluations_nowait ()
excecut e a nonblocking dynamic schedule in a peer partition

void asynchronous_local_evaluations_nowait (PRPQueue &prp_queue)
launch new jobs in prp_queue asynchronously (if capacity is available), perform nonblocking query of all running jobs, and process any completed jobs (handles both local master- and local peer-scheduling cases)

void broadcast_evaluation (const ParamResponsePair &pair)
convenience function for broadcasting an evaluation over an evalComm

void broadcast_evaluation (int fn_eval_id, const Variables &vars, const ActiveSet &set)
convenience function for broadcasting an evaluation over an evalComm

void send_evaluation (PRPQueueIter &prp_it, size_t buff_index, int server_id, bool peer_flag)
helper function for sending sendBuffers[buff_index] to server

void receive_evaluation (PRPQueueIter &prp_it, size_t buff_index, int server_id, bool peer_flag)
helper function for processing recvBuffers[buff_index] within scheduler

void launch_asynch_local (PRPQueueIter &prp_it)
launch an asynchronous local evaluation

void process_asynch_local (int fn_eval_id)
process a completed asynchronous local evaluation

void process_synch_local (PRPQueueIter &prp_it)
process a completed synchronous local evaluation

void assign_asynch_local_queue (PRPQueue &local_prp_queue, PRPQueueIter &local_prp_iter)
helper function for creating an initial active local queue by launching asynch local jobs from local_prp_queue, as limited by server capacity

void assign_asynch_local_queue_nowait (PRPQueue &local_prp_queue, PRPQueueIter &local_prp_iter)
helper function for updating an active local queue by backfilling asynch local jobs from local_prp_queue, as limited by server capacity

size_t test_local_backfill (PRPQueue &assign_queue, PRPQueueIter &assign_iter)
helper function for testing active asynch local jobs and then backfilling

size_t test Receives_backfill (PRPQueueIter &assign_iter, bool peer_flag)
13.5. APPLICATION INTERFACE CLASS REFERENCE

helper function for testing receive requests and then backfilling jobs

• void serve_evaluations_synch ()
  
  serve the evaluation message passing schedulers and perform one synchronous evaluation at a time

• void serve_evaluations_synch_peer ()
  
  serve the evaluation message passing schedulers and perform one synchronous evaluation at a time as part of the 1st peer

• void serve_evaluations_asynch ()
  
  serve the evaluation message passing schedulers and manage multiple asynchronous evaluations

• void serve_evaluations_asynch_peer ()
  
  serve the evaluation message passing schedulers and perform multiple asynchronous evaluations as part of the 1st peer

• void set_evaluation_communicators (const IntArray &message_lengths)
  
  convenience function for updating the local evaluation partition data following ParallelLibrary::init_evaluation_communicators().

• void set_analysis_communicators ()
  
  convenience function for updating the local analysis partition data following ParallelLibrary::init_analysis_communicators().

• void init_serial_evaluations ()
  
  set concurrent evaluation configuration for serial operations

• void init_serial_analyses ()
  
  set concurrent analysis configuration for serial operations (e.g., for local executions on a dedicated master)

• const ParamResponsePair & get_source_pair (const Variables &target_vars)
  
  convenience function for the continuation approach in manage_failure() for finding the nearest successful "source" evaluation to the failed "target"

• void continuation (const Variables &target_vars, const ActiveSet &set, Response &response, const ParamResponsePair &source_pair, int failed_eval_id)
  
  performs a 0th order continuation method to step from a successful "source" evaluation to the failed "target". Invoked by manage_failure() for failAction == "continuation".

• void common_input_filtering (const Variables &vars)
  
  common input filtering operations, e.g. mesh movement

• void common_output_filtering (Response &response)
  
  common output filtering operations, e.g. data filtering

Private Attributes

• int worldSize
  
  size of MPI_COMM_WORLD

• int worldRank
  
  processor rank within MPI_COMM_WORLD

• int iteratorCommSize
  
  size of iteratorComm

• int iteratorCommRank
  
  processor rank within iteratorComm

• bool ieMessagePass
  
  flag for message passing ate the scheduling level

• int numEvalServers
current number of evaluation servers

- int numEvalServersSpec
  user specification for number of evaluation servers

- int procsPerEvalSpec
  user specification for processors per analysis servers

- bool eaMessagePass
  flag for message passing at ea scheduling level

- int numAnalysisServersSpec
  user spec for number of analysis servers

- int procsPerAnalysisSpec
  user specification for processors per analysis servers

- int lenVarsMessage
  length of a MPIPackBuffer containing a Variables object; computed in Model::init_communicators()

- int lenVarsActSetMessage
  length of a MPIPackBuffer containing a Variables object and an ActiveSet object; computed in Model::init_communicators()

- int lenResponseMessage
  length of a MPIPackBuffer containing a Response object; computed in Model::init_communicators()

- int lenPRPairMessage
  length of a MPIPackBuffer containing a ParamResponsePair object; computed in Model::init_communicators()

- short evalScheduling
  user specification of evaluation scheduling algorithm: {DEFAULT, MASTER, PEER_DYNAMIC, PEER_STATIC} - SCHEDULING. Used for manual overrides of auto-configure logic in ParallelLibrary::resolve_inputs().

- short analysisScheduling
  user specification of analysis scheduling algorithm: {DEFAULT, MASTER, PEER} - SCHEDULING. Used for manual overrides of the auto-configure logic in ParallelLibrary::resolve_inputs().

- int asynchLocalEvalConcurrency
  limits the number of concurrent evaluations in asynchronous local scheduling and specifies hybrid concurrency when message passing

- bool asynchLocalEvalStatic
  whether the asynchronous local evaluations are to be performed with a static schedule (default false)

- BitArray localServerAssigned
  array with one bit per logical "server" indicating whether a job is currently running on the server (used for async local static schedules)

- short interfaceSynchronization
  interface synchronization specification: synchronous (default) or asynchronous

- bool headerFlag
  used by synchronize_nowait to manage header output frequency (since this function may be called many times prior to any completions)

- bool asvControlFlag
  used to manage a user request to deactivate the active set vector control. true = modify the ASV each evaluation as appropriate (default); false = ASV values are static so that the user need not check them on each evaluation.

- bool evalCacheFlag
  used to manage a user request to deactivate the function evaluation cache (i.e., queries and insertions using the data_pairs cache).
• bool nearbyDuplicateDetect
  flag indicating optional usage of tolerance-based duplication detection (less efficient, but helpful when experiencing restart cache misses)
• Real nearbyTolerance
  tolerance value for tolerance-based duplication detection
• bool restartFileFlag
  used to manage a user request to deactivate the restart file (i.e., insertions into write_restart).
• SharedResponseData sharedRespData
  SharedResponseData of associated Response.
• String gradientType
  type of gradients present in associated Response
• String hessianType
  type of Hessians present in associated Response
• IntSet gradMixedAnalyticIds
  IDs of analytic gradients when mixed gradients present.
• IntSet hessMixedAnalyticIds
  IDs of analytic gradients when mixed gradients present.
• ShortArray defaultASV
  the static ASV values used when the user has selected asvControl = off
• String failAction
  mitigation action for captured simulation failures: abort, retry, recover, or continuation
• int failRetryLimit
  limit on the number of retries for the retry failAction
• RealVector failRecoveryFnVals
  the dummy function values used for the recover failAction
• IntResponseMap historyDuplicateMap
  used to bookkeep asynchronous evaluations which duplicate data_pairs evaluations. Map key is evalIdCntr, map value is corresponding response.
• std::map<int, std::pair<PRPQueueHIter, Response>> beforeSynchDuplicateMap
  used to bookkeep evalIdCntr, beforeSynchCorePRPQueue iterator, and response of asynchronous evaluations which duplicate queued beforeSynchCorePRPQueue evaluations
• PRPQueue beforeSynchCorePRPQueue
  used to bookkeep vars/set/response of nonduplicate asynchronous core evaluations. This is the queue of jobs populated by asynchronous map() that is later scheduled in synchronize() or synchronize_nowait().
• PRPQueue beforeSynchAlgPRPQueue
  used to bookkeep vars/set/response of asynchronous algebraic evaluations. This is the queue of algebraic jobs populated by asynchronous map() that is later evaluated in synchronize() or synchronize_nowait().
• PRPQueue asynchLocalActivePRPQueue
  used by nonblocking asynchronous local schedulers to bookkeep active local jobs
• std::map<int, IntSizetPair> msgPassRunningMap
  used by nonblocking message passing schedulers to bookkeep which jobs are running remotely
• int nowaitEvalIdRef
CHAPTER 13. CLASS DOCUMENTATION

fnEvalId reference point for preserving modulo arithmetic-based job assignment in case of peer static nonblocking schedulers

- MPIPackBuffer * sendBuffers
  array of pack buffers for evaluation jobs queued to a server
- MPIUnpackBuffer * recvBuffers
  array of unpack buffers for evaluation jobs returned by a server
- MPI_Request * recvRequests
  array of requests for nonblocking evaluation receives

13.5.1 Detailed Description

Derived class within the interface class hierarchy for supporting interfaces to simulation codes. ApplicationInterface provides an interface class for performing parameter to response mappings using simulation code(s). It provides common functionality for a number of derived classes and contains the majority of all of the scheduling algorithms in DAKOTA. The derived classes provide the specifics for managing code invocations using system calls, forks, direct procedure calls, or distributed resource facilities.

13.5.2 Member Function Documentation

void init_serial ( ) [inline], [protected], [virtual]

DataInterface.cpp defaults of 0 servers are needed to distinguish an explicit user request for 1 server (serialization of a parallelism level) from no user request (use parallel auto-config). This default causes problems when init_communicators() is not called for an interface object (e.g., static scheduling fails in DirectApplicInterface::derived_map() for NestedModel::optionalInterface). This is the reason for this function: to reset certain defaults for interface objects that are used serially.

Reimplemented from Interface.

References ApplicationInterface::init_serial_analyses(), and ApplicationInterface::init_serial_evaluations().

void map ( const Variables & vars, const ActiveSet & set, Response & response, bool asynch_flag = false ) [protected], [virtual]

Provides a "mapping" of variables to responses using a simulation. Protected due to Interface letter-envelope idiom.

The function evaluator for application interfaces. Called from derived_evaluate() and derived_evaluate_nowait() in derived Model classes. If asynch_flag is not set, perform a blocking evaluation (using derived_map()). If asynch_flag is set, add the job to the beforeSynchCorePRPQueue queue for execution by one of the scheduler routines in synchronize() or synchronize_nowait(). Duplicate function evaluations are detected with duplication_detect().

Reimplemented from Interface.

References Response::active_set(), Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asvControlFlag, ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::broadcast_evaluation(), Response::copy(), Interface::coreMappings, Interface::currEvalId, Dakota::data_pairs, ApplicationInterface::defaultASV, ApplicationInterface::derived_map(), ApplicationInterface::duplication_detect(), ApplicationInterface::evalCacheFlag, Interface::evalIdCntr, Interface::fineGrainEvalCounters, Interface::fnGradCounter, Interface::fnHessCounter, Interface::fnLabels, Interface::fnValCounter, Response::function_labels(), Interface::init_algebraic_mappings(), ApplicationInterface::init_default_asv(), Interface::init_evaluation_counters(), Interface::interfaceId, ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, Interface::newEvalIdCntr, Interface::newFnGradCounter, Interface::newFnHessCounter,
13.5. APPLICATIONINTERFACE CLASS REFERENCE

Interface::newFnValCounter, Interface::outputLevel, ApplicationInterface::parallelLib, ActiveSet::request_vector(), Interface::response_mapping(), ApplicationInterface::restartFileFlag, ApplicationInterface::sharedRespData, and ParallelLibrary::write_restart().

**const IntResponseMap & synchronize() [protected], [virtual]**

executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns all jobs

This function provides blocking synchronization for all cases of asynchronous evaluations, including the local asynchronous case (background system call, nonblocking fork, & multithreads), the message passing case, and the hybrid case. Called from derived_synchronize() in derived Model classes.

Reimplemented from Interface.

References Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::asynchronous_local_evaluations(), ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Interface::cachedResponseMap, Interface::coreMappings, ApplicationInterface::evalScheduling, ApplicationInterface::historyDuplicateMap, Interface::ieDedMasterFlag, ApplicationInterface::ieMessagePass, Interface::interfaceId, Interface::interfaceType, ApplicationInterface::master_dynamic_schedule_evaluations(), Interface::multiProcEvalFlag, Interface::outputLevel, ApplicationInterface::peer_dynamic_schedule_evaluations(), ApplicationInterface::peer_static_schedule_evaluations(), Interface::rawResponseMap, Interface::response_mapping(), and ApplicationInterface::sharedRespData.

**const IntResponseMap & synchronize_nowait() [protected], [virtual]**

executes a nonblocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns a partial set of completed jobs

This function provides nonblocking synchronization for the local asynchronous case and selected nonblocking message passing schedulers. Called from derived_synchronize_nowait() in derived Model classes.

Reimplemented from Interface.

References Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::asynchronous_local_evaluations_nowait(), ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Interface::cachedResponseMap, Interface::coreMappings, ParamResponsePair::eval_id(), ApplicationInterface::evalScheduling, ApplicationInterface::headerFlag, ApplicationInterface::historyDuplicateMap, Interface::ieDedMasterFlag, ApplicationInterface::ieMessagePass, Interface::interfaceId, Interface::interfaceType, Dakota::lookup_by_eval_id(), ApplicationInterface::master_dynamic_schedule_evaluations_nowait(), Interface::multiProcEvalFlag, Interface::outputLevel, ApplicationInterface::peer_dynamic_schedule_evaluations_nowait(), ApplicationInterface::peer_static_schedule_evaluations_nowait(), Interface::rawResponseMap, ParamResponsePair::response(), Interface::response_mapping(), ApplicationInterface::sharedRespData, and Response::update().

**void serve_evaluations() [protected], [virtual]**

run on evaluation servers to serve the iterator master

Invoked by the serve() function in derived Model classes. Passes control to serve_evaluations_synch(), serve_evaluations_asynch(), serve_evaluations_synch_peer(), or serve_evaluations_asynch_peer() according to specified concurrency, partition, and scheduler configuration.

Reimplemented from Interface.

References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::evalServerId, Interface::ieDedMasterFlag, ApplicationInterface::serve_evaluations_asynch(), ApplicationInterface::serve_evaluations_asynch_peer(), ApplicationInterface::serve_evaluations_synch(), and ApplicationInterface::serve_evaluations_synch_peer().
void stop_evaluation_servers() [protected], [virtual]

used by the iterator master to terminate evaluation servers

   This code is executed on the iteratorComm rank 0 processor when iteration on a particular model is complete. It sends a termination signal (tag = 0 instead of a valid fn_eval_id) to each of the slave analysis servers. NOTE: This function is called from the Strategy layer even when in serial mode. Therefore, use iteratorCommSize to provide appropriate fall through behavior.

Reimplemented from Interface.

References ParallelLibrary::bcast_e(), ParallelLibrary::free(), ParallelConfiguration::ie_parallel_Level(), Interfaces::ieDedMasterFlag, ParallelLibrary::isend_ie(), ApplicationInterface::iteratorCommSize, Interfaces::multiProcEvalFlag, ApplicationInterface::numEvalServers, Interface::outputLevel, ParallelLibrary::parallel_configuration(), and ApplicationInterface::parallelLib.

void init_communicators_checks (int max_eval_concurrency) [protected], [virtual]

perform construct-time error checks on the parallel configuration

   Override DirectApplicInterface definition if plug-in to allow batch processing in Plugin{Serial,Parallel}DirectApplicInterface.cpp

   Reimplemented in DirectApplicInterface, SysCallApplicInterface, and ProcessHandleApplicInterface.

   Referenced by ApplicationInterface::init_communicators().

void set_communicators_checks (int max_eval_concurrency) [protected], [virtual]

perform run-time error checks on the parallel configuration

   Override DirectApplicInterface definition if plug-in to allow batch processing in Plugin{Serial,Parallel}DirectApplicInterface.cpp

   Reimplemented in DirectApplicInterface, SerialDirectApplicInterface, ParallelDirectApplicInterface, SysCallApplicInterface, and ProcessHandleApplicInterface.

   Referenced by ApplicationInterface::set_communicators().

void master_dynamic_schedule_analyses() [protected]

blocking dynamic schedule of all analyses within a function evaluation using message passing

   This code is called from derived classes to provide the master portion of a master-slave algorithm for the dynamic scheduling of analyses among slave servers. It is patterned after master_dynamic_schedule_evaluations(). It performs no analyses locally and matches either serve_analyses_synch() or serve_analyses_asynch() on the slave servers, depending on the value of asynchLocalAnalysisConcurrency. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalAnalysisConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

   References ApplicationInterface::asynchLocalAnalysisConcurrency, ParallelLibrary::free(), ParallelLibrary::irecv_ea(), ParallelLibrary::isend_ea(), ApplicationInterface::numAnalysisDrivers, ApplicationInterface::numAnalysisServers, ApplicationInterface::parallelLib, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

   Referenced by ProcessHandleApplicInterface::create_evaluation_process(), SysCallApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().

void serve_analyses_synch() [protected]

serve the master analysis scheduler and manage one synchronous analysis job at a time
This code is called from derived classes to run synchronous analyses on slave processors. The slaves receive requests (blocking receive), do local derived_map_ac’s, and return codes. This is done continuously until a termination signal is received from the master. It is patterned after serve_evaluations_synch().

References ApplicationInterface::analysisCommRank, ParallelLibrary::bcast_a(), ParallelLibrary::isend_ea(), ApplicationInterface::multiProcAnalysisFlag, ApplicationInterface::parallelLib, ParallelLibrary::recv_ea(), ApplicationInterface::synchronous_local_analysis(), and ParallelLibrary::wait().

Referenced by ProcessHandleApplicInterface::create_evaluation_process(), SysCallApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().

bool duplication_detect ( const Variables & vars, Response & response, bool asynch_flag ) [private]

checks data_pairs and beforeSynchCorePRPQueue to see if the current evaluation request has already been performed or queued.

Called from map() to check incoming evaluation request for duplication with content of data_pairs and beforeSynchCorePRPQueue. If duplication is detected, return true, else return false. Manage bookkeeping with historyDuplicateMap and beforeSynchDuplicateMap. Note that the list searches can get very expensive if a long list is searched on every new function evaluation (either from a large number of previous jobs, a large number of pending jobs, or both). For this reason, a user request for deactivation of the evaluation cache results in a complete bypass of duplication_detect(), even though a beforeSynchCorePRPQueue search would still be meaningful. Since the intent of this request is to streamline operations, both list searches are bypassed.

References Response::active_set(), ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Response::copy(), Dakota::data_pairs, ParamResponsePair::eval_id(), Interface::evalIdCntr, ApplicationInterface::historyDuplicateMap, Interface::interfaceId, Dakota::lookup_by_val(), ApplicationInterface::nearbyDuplicateDetect, ApplicationInterface::nearbyTolerance, and Response::update().

Referenced by ApplicationInterface::map().

void init_default_asv ( size_t num_fns ) [private]

initialize default ASV if needed; this is done at run time due to post-construct time Response size changes.

If the user has specified active_set_vector as off, then map() uses a default ASV which is constant for all function evaluations (so that the user need not check the content of the ASV on each evaluation). Only initialized if needed and not already sized.

References ApplicationInterface::asvControlFlag, ApplicationInterface::defaultASV, ApplicationInterface::gradientType, ApplicationInterface::gradMixedAnalyticIds, ApplicationInterface::hessianType, and ApplicationInterface::hessMixedAnalyticIds.

Referenced by ApplicationInterface::map().

void master_dynamic_schedule_evaluations ( ) [private]

blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a dedicated master partition; executes on iteratorComm master.

This code is called from synchronize() to provide the master portion of a master-slave algorithm for the dynamic scheduling of evaluations among slave servers. It performs no evaluations locally and matches either serve_evaluations_synch() or serve_evaluations_asynch() on the slave servers, depending on the value of asynchLocalEvalConcurrency. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalEvalConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed and returned. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary, peer.
void peer_static_schedule_evaluations( ) [private]

blocking static schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synchronize() in order to manage a static schedule for cases where peer 1 must block when evaluating its local job allocation (e.g., single or multiprocessor direct interface evaluations). It matches serve_evaluations_peer() for any other processors within the first evaluation partition and serve_evaluations_\{synch,asynch\}() for all other evaluation partitions (depending on asynchLocalEvalConcurrency). It performs function evaluations locally for its portion of the job allocation using either asynchronous\_local\_evaluations() or synchronous\_local\_evaluations(). Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary. The iteratorCommRank 0 processor assigns the static schedule since it is the only processor with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchronous\_local\_evaluations(), ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::parallelLib, ApplicationInterface::receive_evaluation(), ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::synchronous\_local\_evaluations(), and ParallelLibrary::waitall().

Referenced by ApplicationInterface::synchronize().

void peer_dynamic_schedule_evaluations( ) [private]

blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synchronize() in order to manage a dynamic schedule, as enabled by nonblocking management of local asynchronous jobs. It matches serve_evaluations_\{synch,asynch\}() for other evaluation partitions, depending on asynchLocalEvalConcurrency; it does not match serve_evaluations_peer() since, for local asynchronous jobs, the first evaluation partition cannot be multiprocessor. It performs function evaluations locally for its portion of the job allocation using asynchronous\_local\_evaluations_nowait(). Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

References ApplicationInterface::assign\_asynch\_local\_queue(), ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::test_local_backfill(), and ApplicationInterface::test\_receives\_backfill().

Referenced by ApplicationInterface::synchronize().

void asynchronous\_local\_evaluations( PRPQueue & local\_prp\_queue ) [private]

perform all jobs in prp_queue using asynchronous approaches on the local processor
This function provides blocking synchronization for the local async case (background system call, nonblocking fork, or threads). It can be called from `synchronize()` for a complete local scheduling of all asynchronous jobs or from `peer_{static,dynamic}_schedule_evaluations()` to perform a local portion of the total job set. It uses `derived_map_asynch()` to initiate asynchronous evaluations and `wait_local_evaluations()` to capture completed jobs, and mirrors the `master_dynamic_schedule_evaluations()` message passing scheduler as much as possible (`wait_local_evaluations()` is modeled after MPI_Waitsome()).

References ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::completionSet, ApplicationInterface::launch_asynch_local(), ApplicationInterface::localServerAssigned, Dakota::lookup_by_eval_id(), ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::process_asynch_local(), Interface::rawResponseMap, and ApplicationInterface::wait_local_evaluations().

Referenced by ApplicationInterface::peer_static_schedule_evaluations(), and ApplicationInterface::synchronize().

```cpp
void synchronous_local_evaluations ( PRPQueue & local_prp_queue ) [private]
perform all jobs in prp_queue using synchronous approaches on the local processor
```

This function provides blocking synchronization for the local synchronous case (foreground system call, blocking fork, or procedure call from `derived_map()`). It is called from `peer_static_schedule_evaluations()` to perform a local portion of the total job set.

References ApplicationInterface::broadcast_evaluation(), Interface::currEvalId, ApplicationInterface::derived_map(), ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, and ApplicationInterface::process_synch_local().

Referenced by ApplicationInterface::peer_static_schedule_evaluations(), and ApplicationInterface::peer_static_schedule_evaluations_nowait().

```cpp
void master_dynamic_schedule_evaluations_nowait ( ) [private]
execute a nonblocking dynamic schedule in a master-slave partition
```

This code is called from `synchronize_nowait()` to provide the master portion of a nonblocking master-slave algorithm for the dynamic scheduling of evaluations among slave servers. It performs no evaluations locally and matches either `serve_evaluations_synch()` or `serve_evaluations_asynch()` on the slave servers, depending on the value of `asynchLocalEvalConcurrency`. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to `asynchLocalEvalConcurrency`). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within `ParallelLibrary`.

References Dakota::abort_handler(), ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::headerFlag, ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synchronize_nowait().

```cpp
void peer_static_schedule_evaluations_nowait ( ) [private]
execute a nonblocking static schedule in a peer partition
```

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from `synchronize_nowait()` in order to manage a nonblocking static schedule. It matches `serve_evaluations_synch()` for other evaluation partitions (`asynchLocalEvalConcurrency == 1`). It performs blocking local function evaluations, one at a time, for its portion of the static schedule and checks for remote completions in between each local completion. Therefore, unlike `peer_dynamic_schedule_evaluations_nowait()`, this scheduler will always return at least one job. Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive, with specific syntax as
encapsulated within *ParallelLibrary*. The iteratorCommRank 0 processor assigns the static schedule since it is
the only processor with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The
alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution
of this function (and therefore the job list) were distributed.

References Dakota::abort_handler(), ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::headerFlag, Interface::interfaceType, Dakota::lookup_by_eval_id(), ApplicationInterface::msgPassRunningMap, Interface::multiProcEvalFlag, ApplicationInterface::nowaitEvalIdRef, ApplicationInterface::numEvalServers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::synchronous_evaluations(), ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synchronize_nowait().

```cpp
void peer_dynamic_schedule_evaluations_nowait( ) [private]
```

execute a nonblocking dynamic schedule in a peer partition

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synchronize_nowait() in
order to manage a nonblocking static schedule. It matches serve_evaluations_{synch,asynch}() for other evaluation
partitions (depending on asynchLocalEvalConcurrency). It performs nonblocking local function evaluations
for its portion of the static schedule using asynchronous_local_evaluations(). Single-level and multilevel parallel
use intra- and inter-communicators, respectively, for send/receive, with specific syntax as encapsulated within
*ParallelLibrary*. The iteratorCommRank 0 processor assigns the dynamic schedule since it is the only processor
with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The alternate design of
each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and
therefore the job list) were distributed.

References Dakota::abort_handler(), ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::headerFlag, Dakota::lookup_by_eval_id(), ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synchronize_nowait().

```cpp
void asynchronous_local_evaluations_nowait ( PRPQueue & local_prp_queue ) [private]
```

launch new jobs in prp_queue asynchronously (if capacity is available), perform nonblocking query of all running
jobs, and process any completed jobs (handles both local master- and local peer-scheduling cases)

This function provides nonblocking synchronization for the local asynchronous case (background system call, non-
blocking fork, or threads). It is called from synchronize_nowait() and passed the complete set of all asynchronous
jobs (beforeSynchCorePRPQueue). It uses derived_map_asynch() to initiate asynchronous evaluations and test-
_local_evaluations() to capture completed jobs in nonblocking mode. It mirrors a nonblocking message passing
scheduler as much as possible (test_local_evaluations() modeled after MPI_Testsome()). The result of this function
is rawResponseMap, which uses eval_id as a key. It is assumed that the incoming local_prp_queue contains only
active and new jobs - i.e., all completed jobs are cleared by synchronize_nowait().

Also supports asynchronous local evaluations with static scheduling. This scheduling policy specifically en-
sures that a completed asynchronous evaluation eval_id is replaced with an equivalent one, modulo asynchEval-
EvalConcurrency. In the nowait case, this could render some servers idle if evaluations don’t come in eval_id
order or some evaluations are cancelled by the caller in between calls. If this function is called with unlimited
local eval concurrency, the static scheduling request is ignored.

References ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActive-
PRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchLocalEvalStatic, Application-
Interface::headerFlag, and ApplicationInterface::test_local_backfill().

Referenced by ApplicationInterface::synchronize_nowait().

void serve_evaluations_synch( ) [private]
serve the evaluation message passing schedulers and perform one synchronous evaluation at a time

This code is invoked by serve_evaluations() to perform one synchronous job at a time on each slave/peer
server. The servers receive requests (blocking receive), do local synchronous maps, and return results. This is
done continuously until a termination signal is received from the master (sent via stop_evaluation_servers()).

References ParallelLibrary::bcast_e(), Interface::currEvalId, ApplicationInterface::derived_map(), Application-
Interface::evalCommRank, ParallelLibrary::isend_ie(), ApplicationInterface::lenResponseMessage, Application-
Interface::lenVarsActSetMessage, ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, Application-
Interface::parallelLib, ParallelLibrary::recv_ie(), MPIPackBuffer::reset(), ApplicationInterface::sharedRespData,
and ParallelLibrary::wait().

Referenced by ApplicationInterface::serve_evaluations().

void serve_evaluations_synch_peer( ) [private]
serve the evaluation message passing schedulers and perform one synchronous evaluation at a time as part of the
1st peer

This code is invoked by serve_evaluations() to perform a synchronous evaluation in coordination with the
iteratorCommRank 0 processor (the iterator) for static schedules. The bcast() matches either the bcast() in
synchronous_local_evaluations(), which is invoked by peer_static_schedule_evaluations(), or the bcast() in map().

References ParallelLibrary::bcast_e(), Interface::currEvalId, ApplicationInterface::derived_map(), Application-
Interface::lenVarsActSetMessage, ApplicationInterface::manage_failure(), ApplicationInterface::parallelLib, and
ApplicationInterface::sharedRespData.

Referenced by ApplicationInterface::serve_evaluations().

void serve_evaluations_asynch( ) [private]
serve the evaluation message passing schedulers and manage multiple asynchronous evaluations

This code is invoked by serve_evaluations() to perform multiple asynchronous jobs on each slave/peer server.
The servers test for any incoming jobs, launch any new jobs, process any completed jobs, and return any results.
Each of these components is nonblocking, although the server loop continues until a termination signal is received
from the master (sent via stop_evaluation_servers()). In the master-slave case, the master maintains the correct
number of jobs on each slave. In the static scheduling case, each server is responsible for limiting concurrency
(since the entire static schedule is sent to the peers at start up).

References Dakota::abort_handler(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface-
::asynchLocalEvalConcurrency, ParallelLibrary::bcast_e(), ApplicationInterface::completionSet, ApplicationInterface-::derived_map_asynch(), ApplicationInterface::evalCommRank, Interface::interfaceId, ParallelLibrary::irecv_ie(),
ApplicationInterface::lenResponseMessage, ApplicationInterface::lenVarsActSetMessage, Dakota::lookup_by_eval-
_id(), Interface::multiProcEvalFlag, ApplicationInterface::parallelLib, ParallelLibrary::recv_ie(), MPIUnpackBuffer-
::reset(), ParallelLibrary::send_ie(), ApplicationInterface::sharedRespData, ParallelLibrary::test(), and Application-
Interface::test_local_evaluations().

Referenced by ApplicationInterface::serve_evaluations().
**CHAPTER 13. CLASS DOCUMENTATION**

```c
void serve_evaluations_asynch_peer() [private]
```

Serve the evaluation message passing schedulers and perform multiple asynchronous evaluations as part of the 1st peer.

This code is invoked by `serve_evaluations()` to perform multiple asynchronous jobs on multiprocessor slave/peer servers. It matches the multiProcEvalFlag bcasts in `ApplicationInterface::asynchronous_local_evaluations()`.

References Dakota::abort_handler(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ParallelLibrary::bcast(), ApplicationInterface::completionSet, ApplicationInterface::derived_map_asynch(), Interface::interfaceId, ApplicationInterface::lenVarsActSetMessage, Dakota::lookup_by_eval_id(), ApplicationInterface::parallelLib, MPIUnpackBuffer::reset(), ApplicationInterface::sharedRespData, and ApplicationInterface::test_local_evaluations().

Referenced by ApplicationInterface::serve_evaluations().

The documentation for this class was generated from the following files:

- ApplicationInterface.hpp
- ApplicationInterface.cpp

### 13.6 Approximation Class Reference

Base class for the approximation class hierarchy.

Inheritance diagram for Approximation:

```
Approximation
| GaussProcApproximation
| PecosApproximation
| SurfpackApproximation
| TANA3Approximation
| TaylorApproximation
| VPSApproximation
```

**Public Member Functions**

- **Approximation ()**
  
  *default constructor*

- **Approximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  *standard constructor for envelope*

- **Approximation (const SharedApproxData &shared_data)**
  
  *alternate constructor*

- **Approximation (const Approximation &approx)**
  
  *copy constructor*

- **virtual ~Approximation ()**
  
  *destructor*

- **Approximation operator= (const Approximation &approx)**
  
  *assignment operator*

- **virtual void build (size_t index=NPOS)**
  
  *builds the approximation from scratch*

- **virtual void export_model (const String &fn_label="", const String &export_prefix="", const unsigned short export_format=NO_MODEL_FORMAT)**
  
  *exports the approximation*
• virtual void rebuild (size_t index=_NPOS)
  rebuilds the approximation incrementally
• virtual void pop (bool save_data)
  removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)
• virtual void push ()
  restores state prior to previous pop()
• virtual void finalize ()
  finalize approximation by applying all remaining trial sets
• virtual void store (size_t index=_NPOS)
  store current approximation state for later combination
• virtual void restore (size_t index=_NPOS)
  restore previous approximation state
• virtual void remove_stored (size_t index=_NPOS)
  remove a stored approximation prior to combination
• virtual void clear_stored ()
  clear stored approximation data
• virtual void combine (size_t swap_index)
  combine current approximation with previously stored approximation
• virtual Real value (const Variables &vars)
  retrieve the approximate function value for a given parameter vector
• virtual const RealVector & gradient (const Variables &vars)
  retrieve the approximate function gradient for a given parameter vector
• virtual const RealSymMatrix & hessian (const Variables &vars)
  retrieve the approximate function Hessian for a given parameter vector
• virtual Real prediction_variance (const Variables &vars)
  retrieve the variance of the predicted value for a given parameter vector
• virtual Real value (const RealVector &c_vars)
  retrieve the approximate function value for a given parameter vector
• virtual const RealVector & gradient (const RealVector &c_vars)
  retrieve the approximate function gradient for a given parameter vector
• virtual const RealSymMatrix & hessian (const RealVector &c_vars)
  retrieve the approximate function Hessian for a given parameter vector
• virtual Real prediction_variance (const RealVector &c_vars)
  retrieve the variance of the predicted value for a given parameter vector
• virtual bool diagnostics_available ()
  check if diagnostics are available for this approximation type
• virtual Real diagnostic (const String &metric,type)
  retrieve a single diagnostic metric for the diagnostic type specified
• virtual RealArray cv_diagnostic (const StringArray &metric_types, unsigned num_folds)
  retrieve diagnostic metrics for the diagnostic types specified, applying
• virtual void primary_diagnostics (int fn_index)
  compute and print all requested diagnostics and cross-validation
• virtual RealArray `challenge_diagnostics` (const StringArray &metric_types, const RealMatrix &challenge_points, const RealVector &challenge_responses)
  compute requested diagnostics for user provided challenge pts
• virtual void `challenge_diagnostics` (int fn_index, const RealMatrix &challenge_points, const RealVector &challenge_responses)
  compute and print all requested diagnostics for user provided challenge pts
• virtual RealVector `approximation_coefficients` (bool normalized) const
  return the coefficient array computed by `build()|rebuild()`
• virtual void `approximation_coefficients` (const RealVector &approx_coeffs, bool normalized)
  set the coefficient array from external sources, rather than computing with `build()|rebuild()`
• virtual void `coefficient_labels` (std::vector<std::string> &coeff_labels) const
  print the coefficient array computed in `build()|rebuild()`
• virtual void `print_coefficients` (std::ostream &s, bool normalized)
  print the coefficient array computed in `build()|rebuild()`
• virtual int `min_coefficients` () const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in numVars dimensions
• virtual int `recommended_coefficients` () const
  return the recommended number of samples (unknowns) required to build the derived class approximation type in numVars dimensions
• virtual int `num_constraints` () const
  return the number of constraints to be enforced via an anchor point
• virtual void `clear_current` ()
  clear current build data in preparation for next build
• int `min_points` (bool constraint_flag) const
  return the minimum number of points required to build the approximation type in numVars dimensions. Uses `*_coefficients()` and `num_constraints()`.
• int `recommended_points` (bool constraint_flag) const
  return the recommended number of points required to build the approximation type in numVars dimensions (default same as `min_points`)
• const Pecos::SurrogateData & `approximation_data` () const
  return approxData
• void `add` (const Pecos::SurrogateDataVars &sdv, bool anchor_flag)
  append to SurrogateData::varsData or assign to SurrogateData::anchorVars
• void `add` (const Variables &vars, bool anchor_flag, bool deep_copy)
  extract the relevant vectors from Variables and invoke add(RealVector&), IntVector&, RealVector&)
• void `add` (const Real *sample_c_vars, bool anchor_flag, bool deep_copy)
  create a RealVector view and invoke add(RealVector&, empty, empty)
• void `add` (const RealVector &c_vars, const IntVector &di_vars, const RealVector &dr_vars, bool anchor_flag, bool deep_copy)
  shared code among add(Variables&) and add(Real*); adds a new data point by either appending to SurrogateData::varsData or assigning to SurrogateData::anchorVars, as dictated by anchor_flag. Uses add_point() and add_anchor().
• void `add` (const Pecos::SurrogateDataResp &sdr, bool anchor_flag)
### 13.6. APPROXIMATION CLASS REFERENCE

- **void add (const Response &response, int fn_index, bool anchor_flag, bool deep_copy)**
  
  adds a new data point by either appending to SurrogateData::respData or assigning to SurrogateData::anchorResp, as dictated by anchor_flag. Uses add_point() and add_anchor().

- **void add (const RealMatrix &sample_vars, const RealVector &sample_resp)**
  
  add data from the provided samples and response matrices, assuming continuous variables and function values only.

- **void pop_count (size_t count)**
  
  appends to SurrogateData::popCountStack (number of entries to pop from end of SurrogateData::{vars,resp}Data, based on size of last data append)

- **void clear_all ()**
  
  end of SurrogateData::{vars,resp}Data, based on size of last data append

- **void clear_anchor ()**
  
  clear SurrogateData::anchor{Vars,Resp}

- **void clear_data ()**
  
  clear SurrogateData::{vars,resp}Data

- **void clear_popped ()**
  
  clear SurrogateData::popped{Vars,Resp}Trials,popCountStack

- **void set_bounds (const RealVector &c_l_bnds, const RealVector &c_u_bnds, const IntVector &d_l_bnds, const IntVector &d_u_bnds, const RealVector &d_r_l_bnds, const RealVector &d_r_u_bnds)**
  
  set approximation lower and upper bounds (currently only used by graphics)

- **Approximation * approx_rep () const**
  
  returns approxRep for access to derived class member functions that are not mapped to the top Approximation level

### Protected Member Functions

- **Approximation (BaseConstructor, const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Approximation (NoDBBaseConstructor, const SharedApproxData &shared_data)**
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **void check_points (size_t num_build_pts)**
  
  Check number of build points against minimum required.

### Protected Attributes

- **RealVector approxGradient**
  
  gradient of the approximation returned by gradient()

- **RealSymMatrix approxHessian**
  
  Hessian of the approximation returned by hessian()

- **Pecos::SurrogateData approxData**
  
  contains the variables/response data for constructing a single approximation model (one response function)

- **SharedApproxData * sharedDataRep**
contains the approximation data that is shared among the response set

- String approxLabel

  label for approximation, if applicable

### Private Member Functions

- **Approximation * get_approx (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**

  Used only by the standard envelope constructor to initialize approxRep to the appropriate derived type.

- **Approximation * get_approx (const SharedApproxData &shared_data)**

  Used only by the alternate envelope constructor to initialize approxRep to the appropriate derived type.

### Private Attributes

- **Approximation * approxRep**

  pointer to the letter (initialized only for the envelope)

- **int referenceCount**

  number of objects sharing approxRep

### 13.6.1 Detailed Description

Base class for the approximation class hierarchy.

The **Approximation** class is the base class for the response data fit approximation class hierarchy in DAKOTA. One instance of an Approximation must be created for each function to be approximated (a vector of Approximations is contained in ApproximationInterface). For memory efficiency and enhanced polymorphism, the approximation hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Approximation) serves as the envelope and one of the derived classes (selected in Approximation::get_approx()) serves as the letter.

### 13.6.2 Constructor & Destructor Documentation

**Approximation ( )**

default constructor

The default constructor is used in Array<Approximation> instantiations and by the alternate envelope constructor. approxRep is NULL in this case (problem_db is needed to build a meaningful Approximation object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

**Approximation ( ProblemDescDB & problem_db, const SharedApproxData & shared_data, const String & approx_label )**

standard constructor for envelope

Envelope constructor only needs to extract enough data to properly execute get_approx, since Approximation(-BaseConstructor, problem_db) builds the actual base class data for the derived approximations. References Dakota::abort_handler(), Approximation::approxRep, and Approximation::get_approx().
13.6. APPROXIMATION CLASS REFERENCE

Approximation (const SharedApproxData & shared_data)

alternate constructor
This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to problem_db, it utilizes the NoDBBaseConstructor constructor chain.
References Dakota::abort_handler(), Approximation::approxRep, and Approximation::get_approx().

Approximation (const Approximation & approx)
copy constructor
Copy constructor manages sharing of approxRep and incrementing of referenceCount.
References Approximation::approxRep, and Approximation::referenceCount.

~Approximation() [virtual]
destructor
Destructor decrements referenceCount and only deletes approxRep when referenceCount reaches zero.
References Approximation::approxRep, and Approximation::referenceCount.

Approximation (BaseConstructor, const ProblemDescDB & problem_db, const SharedApproxData & shared_data, const String & approx_label) [protected]
constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all derived classes. get_approx() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_approx() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Approximation).

Approximation (NoDBBaseConstructor, const SharedApproxData & shared_data) [protected]
constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all derived classes. get_approx() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_approx() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Approximation).

13.6.3 Member Function Documentation

Approximation operator=(const Approximation & approx)
assignment operator
References Approximation::approxRep, and Approximation::referenceCount.
void build ( size_t index = _NPOS ) [virtual]

builds the approximation from scratch
This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.
Reimplemented in PecosApproximation, VPSApproximation, GaussProcApproximation, SurfpackApproximation, TaylorApproximation, and TANA3Approximation.
References Approximation::approxData, Approximation::approxRep, Approximation::build(), and Approximation::check_points().
Referred by TANA3Approximation::build(), TaylorApproximation::build(), Approximation::build(), SurfpackApproximation::build(), GaussProcApproximation::build(), VPSApproximation::build(), PecosApproximation::build(), and Approximation::rebuild().

void export_model ( const String & fn_label = "", const String & export_prefix = "", const unsigned short export_format = NO_MODEL_FORMAT ) [virtual]

exports the approximation
This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.
Reimplemented in SurfpackApproximation.
References Approximation::approxRep, and Approximation::export_model().
Referred by Approximation::export_model().

void rebuild ( size_t index = _NPOS ) [virtual]

rebuilds the approximation incrementally
This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.
Reimplemented in PecosApproximation.
References Approximation::approxRep, Approximation::build(), and Approximation::rebuild().
Referred by Approximation::rebuild().

void pop ( bool save_data ) [virtual]

removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)
This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.
Reimplemented in PecosApproximation.
References Approximation::approxData, Approximation::approxRep, and Approximation::pop().
Referred by Approximation::pop(), and PecosApproximation::pop().

void push ( ) [virtual]

restores state prior to previous pop()
This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.
Reimplemented in PecosApproximation.
References Approximation::approxData, Approximation::approxRep, Approximation::push(), SharedApproxData::retrieval_index(), and Approximation::sharedDataRep.
Referred by Approximation::push(), and PecosApproximation::push().
void finalize ( ) [virtual]
finalize approximation by applying all remaining trial sets
   This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.
   Reimplemented in PecosApproximation.
   References Approximation::approxData, Approximation::approxRep, SharedApproxData::finalization_index(), Approximation::finalize(), and Approximation::sharedDataRep.
   Referenced by Approximation::finalize(), and PecosApproximation::finalize().

void clear_current ( ) [inline], [virtual]
clear current build data in preparation for next build
   Redefined by TANA3Approximation to clear current data but preserve history.
   Reimplemented in TANA3Approximation.
   References Approximation::approxRep, Approximation::clear_all(), and Approximation::clear_current().
   Referenced by Approximation::clear_all().

void clear_all ( ) [inline]
end of SurrogateData::{vars,resp}Data, based on size of last data append)
clear all build data (current and history) to restore original state
   Clears out any history (e.g., TANA3Approximation use for a different response function in NonDReliability).
   References Approximation::approxData, Approximation::approxRep, and Approximation::clear_all().
   Referenced by Approximation::clear_all(), and Approximation::clear_current().

Approximation * get_approx ( ProblemDescDB & problem_db, const SharedApproxData & shared_data, const String & approx_label ) [private]
Used only by the standard envelope constructor to initialize approxRep to the appropriate derived type.
   Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
   References SharedApproxData::approxType, SharedApproxData::data_rep(), ProblemDescDB::get_bool(), and Dakota::strends().
   Referenced by Approximation::Approximation().

Approximation * get_approx ( const SharedApproxData & shared_data ) [private]
Used only by the alternate envelope constructor to initialize approxRep to the appropriate derived type.
   Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
   References SharedApproxData::approxType, SharedApproxData::data_rep(), and Dakota::strends().
The documentation for this class was generated from the following files:
   • DakotaApproximation.hpp
   • DakotaApproximation.cpp

13.7  ApproximationInterface Class Reference
Derived class within the interface class hierarchy for supporting approximations to simulation-based results.
   Inheritance diagram for ApproximationInterface:
Public Member Functions

- **ApproximationInterface** (ProblemDescDB &problem_db, const Variables &am_vars, bool am_cache, const String &am_interface_id, const StringArray &fn_labels)
  
  *primary constructor*

- **ApproximationInterface** (const String &approx_type, const UShortArray &approx_order, const Variables &am_vars, bool am_cache, const String &am_interface_id, size_t num_fns, short data_order, short output_level)
  
  *alternate constructor for instantiations on the fly*

- ∼ApproximationInterface ()
  
  *destructor*

Protected Member Functions

- void map (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)
  
  *the function evaluator: provides an approximate “mapping” from the variables to the responses using function-Surfaces*

- int minimum_points (bool constraint_flag) const
  
  *returns the minimum number of samples required to build the functionSurfaces*

- int recommended_points (bool constraint_flag) const
  
  *returns the recommended number of samples recommended to build the functionSurfaces*

- void approximation_function_indices (const IntSet &approx_fn_indices)
  
  *set the (currently active) approximation function index set*

- void update_approximation (const Variables &vars, const IntResponsePair &response_pr)

- void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)

- void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)

- void append_approximation (const Variables &vars, const IntResponsePair &response_pr)

- void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)

- void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)

- void build_approximation (const RealVector &c_lbnds, const RealVector &c_ubnds, const IntVector &di_lbnds, const IntVector &di_u_bnds, const IntVector &di_u_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds, size_t index=NPOS)

- void export_approximation ()

- void rebuild_approximation (const BoolDeque &rebuild_deque, size_t index=NPOS)

- void pop_approximation (bool save_surr_data)

- void push_approximation ()

- bool push_available ()
  
  *queries the approximation for the ability to retrieve a previous increment*

- void finalize_approximation ()
13.7. APPROXIMATIONINTERFACE CLASS REFERENCE

finalizes the approximation by applying all trial increments

- void `store_approximation(size_t index=_NPOS)`
  move the current approximation into storage for later combination; the index of the stored approximation can be passed to allow replacement instead of augmentation (default is `push_back`)

- void `restore_approximation(size_t index=_NPOS)`
  return an approximation from storage; the index identifies a particular stored data set (default is `pop_back` from stored)

- void `remove_stored_approximation(size_t index=_NPOS)`
  remove a stored approximation, due to redundancy with the current approximation, prior to combination (default for no index is `pop_back`)

- void `clear_stored()`
  clear stored approximation data

- void `combine_approximation()`
  combine the current approximation with previously stored data sets

- Real2DArray `cv_diagnostics` (const StringArray &metrics, unsigned num_folds)
  approximation cross-validation quality metrics per response function

- Real2DArray `challenge_diagnostics` (const StringArray &metric_types, const RealMatrix &challenge_pts, const RealVector &challenge_resps)

- void `clear_current()`
  clears current data from an approximation interface

- void `clear_all()`
  clears all data from an approximation interface

- void `clear_popped()`
  clears bookkeeping for popped data sets from an approximation interface

- SharedApproxData & `shared_approximation()`
  retrieve the `SharedApproxData` within an `ApproximationInterface`

- std::vector< Approximation > & `approximations()`
  retrieve the `Approximations` within an `ApproximationInterface`

- const Pecos::SurrogateData & `approximation_data(size_t index)`
  retrieve the approximation data from a particular `Approximation` within an `ApproximationInterface`

- const RealVectorArray & `approximation_coefficients(bool normalized=false)`
  retrieve the approximation coefficients from each `Approximation` within an `ApproximationInterface`

- void `approximation_coefficients(const RealVectorArray &approx_coeffs, bool normalized=false)`
  set the approximation coefficients within each `Approximation` within an `ApproximationInterface`

- const RealVector & `approximation_variances(const Variables &vars)`
  retrieve the approximation variances from each `Variables` within an `ApproximationInterface`

- const IntResponseMap & `synchronize()`
  recovers data from a series of asynchronous evaluations (blocking)

- const IntResponseMap & `synchronize_nowait()`
  recovers data from a series of asynchronous evaluations (nonblocking)
CHAPTER 13. CLASS DOCUMENTATION

Private Member Functions

- void mixed_add (const Variables &vars, const Response &response, bool anchor)
  add variables/response data to functionSurfaces using a mixture of shallow and deep copies
- void mixed_add (const Real *c_vars, const Response &response, bool anchor)
  add variables/response data to functionSurfaces using a mixture of shallow and deep copies
- void shallow_add (const Variables &vars, const Response &response, bool anchor)
  add variables/response data to functionSurfaces using a shallow copy
- void sample_to_variables (const Real *sample_c_vars, size_t num_cv, Variables &vars)
  populate continuous variables within vars from sample_c_vars
- void update_pop_counts (const IntResponseMap &resp_map)
  append to the stack of pop counts within each of the functionSurfaces based on the active set definitions within
  resp_map
- void read_challenge_points (bool active_only)
  Load approximation test points from user challenge points file.

Private Attributes

- IntSet approxFnIndices
  for incomplete approximation sets, this array specifies the response function subset that is approximated
- SharedApproxData sharedData
  data that is shared among all functionSurfaces
- std::vector< Approximation > functionSurfaces
  list of approximations, one per response function
- RealVectorArray functionSurfaceCoeffs
  array of approximation coefficient vectors, one per response function
- RealVector functionSurfaceVariances
  vector of approximation variances, one value per response function
- String challengeFile
  data file for user-supplied challenge data (per interface, since may contain multiple responses)
- unsigned short challengeFormat
  tabular format of the challenge points file
- bool challengeActiveOnly
  whether to import active only
- RealMatrix challengePoints
  container for the challenge points data (variables only)
- RealMatrix challengeResponses
  container for the challenge points data (responses only)
- Variables actualModelVars
  copy of the actualModel variables object used to simplify conversion among differing variable views
- bool actualModelCache
  indicates usage of an evaluation cache by the actualModel
- String actualModellnterfaceId
  the interface id from the actualModel used for ordered PRPCache lookups
13.7. APPROXIMATIONINTERFACE CLASS REFERENCE

- **IntResponseMap beforeSynchResponseMap**

  bookkeeping map to catalogue responses generated in map() for use in synchronize() and synchronize_nowait().

  This supports pseudo-asynchronous operations (approximate responses are always computed synchronously, but asynchronous virtual functions are supported through bookkeeping).

## Additional Inherited Members

### 13.7.1 Detailed Description

Derived class within the interface class hierarchy for supporting approximations to simulation-based results.

**ApproximationInterface** provides an interface class for building a set of global/local/multipoint approximations and performing approximate function evaluations using them. It contains a list of Approximation objects, one for each response function.

### 13.7.2 Member Function Documentation

#### void update_approximation ( const Variables & vars, const IntResponsePair & response_pr )

[protected], [virtual]

This function populates/replaces each Approximation::anchorPoint with the incoming variables/response data point.

  Reimplemented from Interface.

  References ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

#### void update_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map )

[protected], [virtual]

This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.

  Reimplemented from Interface.

  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::actualModelVars, ApproximationInterface::approxFnIndces, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::sample_to_variables(), and ApproximationInterface::shallow_add().

#### void update_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map )

[protected], [virtual]

This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.

  Reimplemented from Interface.

  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::approxFnIndces, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

#### void append_approximation ( const Variables & vars, const IntResponsePair & response_pr )

[protected], [virtual]

This function appends to each Approximation::currentPoints with one incoming variables/response data point.

  Reimplemented from Interface.
References ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::approxFnIndices, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

```cpp
void append_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map )
[protected], [virtual]
```
This function appends to each Approximation::currentPoints with multiple incoming variables/response data points.
Reimplemented from Interface.
References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::actualModelVars, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::sample_to_variables(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

```cpp
void append_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map )
[protected], [virtual]
```
This function appends to each Approximation::currentPoints with multiple incoming variables/response data points.
Reimplemented from Interface.
References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

```cpp
void build_approximation ( const RealVector & c_lbnds, const RealVector & c_ubnds, const IntVector & d_lbnds, const IntVector & d_ubnds, const RealVector & d_rlbnds, const RealVector & d_rubnds, size_t index = NPOS )
[protected], [virtual]
```
This function finds the coefficients for each Approximation based on the data passed through update_approximation() calls. The bounds are used only for graphics visualization.
Reimplemented from Interface.
References ApproximationInterface::approxFnIndices, SharedApproxData::build(), ApproximationInterface::challengeFile, ApproximationInterface::challengePoints, ApproximationInterface::challengeResponses, ApproximationInterface::functionSurfaces, ApproximationInterface::read_challenge_points(), SharedApproxData::set_bounds(), and ApproximationInterface::sharedData.

```cpp
void export_approximation ( )
[protected], [virtual]
```
This function calls export on each approximation
Reimplemented from Interface.
References ApproximationInterface::approxFnIndices, and ApproximationInterface::functionSurfaces.

```cpp
void rebuild_approximation ( const BoolDeque & rebuild_deque, size_t index = NPOS )
[protected], [virtual]
```
This function updates the coefficients for each Approximation based on data increments provided by {update.append}-approximation().
Reimplemented from Interface.
References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::rebuild(), and ApproximationInterface::sharedData.
13.7. APPROXIMATIONINTERFACE CLASS REFERENCE

void pop_approximation ( bool save_surr_data ) [inline], [protected], [virtual]

This function removes data provided by a previous append_approximation() call, possibly different numbers for each function, or as specified in pop_count, which is assumed to be the same for all functions.

Reimplemented from Interface.

References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::pop(), and ApproximationInterface::sharedData.

void push_approximation ( ) [inline], [protected], [virtual]

This function updates the coefficients for each Approximation based on data increments provided by {update.append}_approximation() (denoted in the text).

Reimplemented from Interface.

References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::post_push(), SharedApproxData::pre_push(), and ApproximationInterface::sharedData.

void read_challenge_points ( bool active_only ) [private]

Load approximation test points from user challenge points file.

Challenge data defaults to active/inactive, but user can override to active only.

References ApproximationInterface::actualModelVars, ApproximationInterface::challengeActiveOnly, ApproximationInterface::challengeFile, ApproximationInterface::challengeFormat, ApproximationInterface::challengePoints, ApproximationInterface::challengeResponses, Variables::copy(), and ApproximationInterface::functionSurfaces.

Referenced by ApproximationInterface::build_approximation().

13.7.3 Member Data Documentation

std::vector<Approximation> functionSurfaces [private]

list of approximations, one per response function

This formulation allows the use of mixed approximations (i.e., different approximations used for different response functions), although the input specification is not currently general enough to support it.

Referenced by ApproximationInterface::append_approximation(), ApproximationInterface::approximation_coefficients(), ApproximationInterface::approximation_data(), ApproximationInterface::approximation_variances(), ApproximationInterface::build_approximation(), ApproximationInterface::clear_all(), ApproximationInterface::clear_current(), ApproximationInterface::clear_popped(), ApproximationInterface::clear_stored(), ApproximationInterface::combine_approximation(), ApproximationInterface::cv_diagnostics(), ApproximationInterface::export_approximation(), ApproximationInterface::finalize_approximation(), ApproximationInterface::map(), ApproximationInterface::minimum_points(), ApproximationInterface::mixed_add(), ApproximationInterface::pop_approximation(), ApproximationInterface::push_approximation(), ApproximationInterface::read_challenge_points(), ApproximationInterface::rebuild_approximation(), ApproximationInterface::recommended_points(), ApproximationInterface::restore_approximation(), ApproximationInterface::shallow_add(), ApproximationInterface::store_approximation(), ApproximationInterface::update_approximation(), and ApproximationInterface::update_pop_counts().

The documentation for this class was generated from the following files:

- ApproximationInterface.hpp
- ApproximationInterface.cpp
13.8 APPSEvalMgr Class Reference

Evaluation manager class for APPSPACK.
Inherits Executor.

Public Member Functions

- `APPSEvalMgr (Optimizer &, Model &model)`
  constructor
- `~APPSEvalMgr ()`
  destructor
- `bool isReadyForWork () const`
tells APPS whether or not there is a processor available to perform a function evaluation
- `bool submit (const int apps_tag, const HOPSPACK::Vector &apps_xtrial, const HOPSPACK::EvalRequest-Type apps_request)`
  performs a function evaluation at APPS-provided x_in
- `int recv (int &apps_tag, HOPSPACK::Vector &apps_f, HOPSPACK::Vector &apps_cEqs, HOPSPACK::-Vector &apps_cIneqs, string &apps_msg)`
  returns a function value to APPS
- `std::string getEvaluatorType (void) const`
  return the type of the Dakota linked evaluator
- `void printDebugInfo (void) const`
  empty implementation of debug info needed to complete the interface
- `void printTimingInfo (void) const`
  empty implementation of timing info needed to complete the interface
- `void set_asynch_flag (const bool dakotaAsynchFlag)`
  publishes whether or not to do asynchronous evaluations
- `void set_blocking_synch (const bool blockingSynchFlag)`
  publishes whether or not APPS is operating synchronously
- `void set_total_workers (const int numDakotaWorkers)`
  publishes the number of processors available for function evaluations

Private Attributes

- `Optimizer & dakOpt`
  reference to the DakotaOptimizer
- `Model & iteratedModel`
  reference to the APPSOptimizer’s model passed in the constructor
- `bool modelAsynchFlag`
  flag for asynchronous function evaluations
- `bool blockingSynch`
  flag for APPS synchronous behavior
- `int numWorkersUsed`
  number of processors actively performing function evaluations
- `int numWorkersTotal`
total number of processors available for performing function evaluations

• RealVector xTrial
  trial iterate

• std::map<int, int> tagList
  map of DAKOTA eval id to APPS eval id (for asynchronous evaluations)

• std::map<int, RealVector> functionList
  map of APPS eval id to responses (for synchronous evaluations)

• IntResponseMap dakotaResponseMap
  map of DAKOTA responses returned by synchronize_nowait()

13.8.1 Detailed Description
Evaluation manager class for APPSPACK.

The APPSEvalMgr class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.

13.8.2 Constructor & Destructor Documentation

APPSEvalMgr ( Optimizer & opt, Model & model )

constructor

Evaluation manager class for APPSPACK.

The APPSEvalMgr class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.

13.8.3 Member Function Documentation

bool isReadyForWork ( ) const

 tells APPS whether or not there is a processor available to perform a function evaluation

 Check to see if all processors available for function evaluations are being used. If not, tell APPS that one is available.

 References APPSEvalMgr::numWorkersTotal, and APPSEvalMgr::numWorkersUsed.

bool submit ( const int apps_tag, const HOPSPACK::Vector & apps_xtrial, const HOPSPACK::EvalRequestType apps_request )

performs a function evaluation at APPS-provided x_in

 Convert APPSPACK vector of variables to DAKOTA vector of variables and perform function evaluation asynchronously or not as specified in the DAKOTA input deck. If evaluation is asynchronous, map the dakota id to the APPS tag. If evaluation is synchronous, map the responses to the APPS tag.

 References Model::current_response(), Model::current_variables(), Model::evaluate(), Model::evaluate_nowait(), Model::evaluation_id(), Response::function_values(), APPSEvalMgr::functionList, APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, APPSEvalMgr::numWorkersTotal, APPSEvalMgr::numWorkersUsed, and APPSEvalMgr::tagList.
int recv ( int & apps_tag, HOPSPACK::Vector & apps_f, HOPSPACK::Vector & apps_cEqs, HOPSPACK::Vector & apps_cIneqs, string & apps_msg )

returns a function value to APPS

Retrieve a set of responses values, convert to APPS data structures, and return them to APPS. APPS tags are tied to corresponding responses using the appropriate (i.e., asynchronous or synchronous) map.

References APPSEvalMgr::blockingSynch, APPSEvalMgr::dakOpt, APPSEvalMgr::dakotaResponseMap, APPSEvalMgr::functionList, Optimizer::get_responses_from_dakota(), APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, APPSEvalMgr::numWorkersUsed, Model::synchronize(), Model::synchronize_nowait(), and APPSEvalMgr::tagList.

The documentation for this class was generated from the following files:

- APPSEvalMgr.hpp
- APPSEvalMgr.cpp

### 13.9 AppsTraits Class Reference

Wrapper class for HOPSPACK.

Inheritance diagram for AppsTraits:

```
          TraitsBase
            |
  AppsTraits
```

#### Public Types

- typedef HOPSPACK::Hopspack OptT
- typedef HOPSPACK::Vector VecT
- typedef HOPSPACK::Matrix MatT

#### Public Member Functions

- **AppsTraits ()**
  
  *default constructor*

- **virtual ~AppsTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**

  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**

  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_discrete_variables ()**

  *Return the flag indicating whether method supports discrete variables.*

- **bool supports_linear_equality ()**

  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
Return the flag indicating whether method supports linear inequalities.

- `bool supports_nonlinear_inequality()`

Return the flag indicating whether method supports nonlinear equalities.

- `NONLINEAR_EQUALITY_FORMAT nonlinear_equality_format()`

Return the format used for nonlinear equality constraints.

- `bool supports_nonlinear_inequality()`

Return the flag indicating whether method supports nonlinear inequalities.

- `NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format()`

Return the format used for nonlinear inequality constraints.

### Static Public Member Functions

- `static double noValue()`  
- `static double getBestObj(const OptT &)`  
- `static void copy_matrix_data(const RealMatrix &source, HOPSPACK::Matrix &target)`

### 13.9.1 Detailed Description

Wrapper class for HOPSPACK.

The APPSOptimizer class provides a wrapper for HOPSPACK, a Sandia-developed C++ library for general- 
ized pattern search. HOPSPACK defaults to a coordinate pattern search but also allows for augmented search 
patterns. It can solve problems with bounds, linear constraints, and general nonlinear constraints. APPSOptimizer 
uses an APPSEvalMgr object to manage the function evaluations.

The user input mappings are as follows: output max function evaluations, constraint tol initial delta, contraction factor, threshold delta, solution target, synchronization, merit function, constraint penalty, and smoothing factor are mapped into HOPS's "Display", "Maximum Evaluations", "Active Tolerance"/"Nonlinear Active Tolerance", "Initial Step", "Contraction Factor", "Step Tolerance", "Objective Target", "Synchronous Evaluations", "Penalty Function", "Penalty Parameter", and "Penalty Smoothing Value" data attributes. Refer to the HOPS web site (https://software.sandia.gov/trac/hopspack) for additional information on HOPS objects and controls. HOPSPACK-specific traits class. AppsTraits specializes some traits accessors by over-riding the default accessors in TraitsBase.

The documentation for this class was generated from the following files:

- `APPSOptimizer.hpp`
- `APPSOptimizer.cpp`

### 13.10 ApreproWriter Class Reference

Utility used in derived write_core to write in aprepro format.

### Public Member Functions

- `template<typename ArrayType> void operator()(std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)`
13.10.1 Detailed Description
Utility used in derived write_core to write in aprepro format.

The documentation for this class was generated from the following file:

- DakotaVariables.hpp

13.11 BaseConstructor Struct Reference
Dummy struct for overloading letter-envelope constructors.

Public Member Functions

- BaseConstructor (int=0)

  C++ structs can have constructors.

13.11.1 Detailed Description
Dummy struct for overloading letter-envelope constructors.

  BaseConstructor is used to overload the constructor for the base class portion of letter objects. It avoids infinite recursion (Coplien p.139) in the letter-envelope idiom by preventing the letter from instantiating another envelope. Putting this struct here avoids circular dependencies.

  The documentation for this struct was generated from the following file:

- dakota_global_defs.hpp

13.12 BootstrapSampler< Data > Class Template Reference
Actual bootstrap sampler implementation for common data types.

Inheritance diagram for BootstrapSampler< Data >:

```
  BootstrapSamplerBase< Data >
  
  BootstrapSampler< Data >
  
  BootstrapSamplerWithGS< Data, Getter, Setter >
```

Public Member Functions

- BootstrapSampler (const Data &orig_data, size_t block_size=1)
  Constructor for the sampler.

- virtual ~BootstrapSampler ()
  Destructor.

- virtual void operator() (size_t num_samp, Data &bootstrapped_sample)
13.13. BOOTSTRAPSAMPLER\langle\text{TEUCHOS}\!:\!\text{SERIALDENSEMATRIX}\langle\text{ORDINALTYPE, SCALARTYPE}\rangle\rangle \rightarrow \text{CLASS TEMPLATE REFERENCE}

Protected Attributes

- size_t blockSize

  Size of the block defining a sample.

Additional Inherited Members

13.12.1 Detailed Description

template\langle\text{typename Data}\rangle \text{class Dakota}\!:\!\text{BootstrapSampler}\langle\text{Data}\rangle

Actual bootstrap sampler implementation for common data types.

Template requires the given type to support an STL-like interface, including a size method and begin and end methods returning random access iterators

13.12.2 Member Function Documentation

virtual void operator() ( size_t num_samp, Data & bootstrapped_sample ) [inline], [virtual]

Implements BootstrapSamplerBase\langle\text{Data}\rangle.

Reimplemented in BootstrapSamplerWithGS\langle\text{Data, Getter, Setter}\rangle.

References BootstrapSampler\langle\text{Data}\rangle::blockSize, BootstrapSamplerBase\langle\text{Data}\rangle::bootstrapRNG, BootstrapSamplerBase\langle\text{Data}\rangle::origData, and BootstrapSamplerBase\langle\text{Data}\rangle::sampler.

The documentation for this class was generated from the following file:

- BootstrapSampler.hpp

13.13 BootstrapSampler\langle\text{Teuchos}\!:\!\text{SerialDenseMatrix}\langle\text{OrdinalType, SCALARTYPE}\rangle\rangle \rightarrow \text{CLASS TEMPLATE REFERENCE}

Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix.

Inheritance diagram for BootstrapSampler\langle\text{Teuchos}\!:\!\text{SerialDenseMatrix}\langle\text{OrdinalType, SCALARTYPE}\rangle\rangle:

\begin{center}
\begin{tabular}{|c|}
\hline
BootstrapSamplerBase\langle\text{Teuchos}\!:\!\text{SerialDenseMatrix}\langle\text{OrdinalType, SCALARTYPE}\rangle\rangle \\
\hline
BootstrapSampler\langle\text{Teuchos}\!:\!\text{SerialDenseMatrix}\langle\text{OrdinalType, SCALARTYPE}\rangle\rangle \\
\hline
\end{tabular}
\end{center}

Public Types

- typedef Teuchos::SerialDenseMatrix\langle\text{OrdinalType, SCALARTYPE}\rangle \text{MatType}

  Convenience definition.
Public Member Functions

- BootstrapSampler (const MatType &orig_data, size_t blockSize=1)
  
  Constructor for the sampler.

- virtual ~BootstrapSampler ()
  
  Destructor.

- virtual void operator() (size_t num_samp, MatType &bootstrapped_sample)

Protected Attributes

- size_t blockSize

  Size of the block defining a sample.

Additional Inherited Members

13.13.1 Detailed Description

template<typename OrdinalType, typename ScalarType>
class Dakota::BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >

Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix.

13.13.2 Member Function Documentation

virtual void operator() ( size_t num_samp, MatType & bootstrapped_sample ) [inline],[virtual]

Implements BootstrapSamplerBase< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >.

References BootstrapSampler< Data >::blockSize, BootstrapSamplerBase< Data >::bootstrapRNG, BootstrapSamplerBase< Data >::origData, and BootstrapSamplerBase< Data >::sampler.

The documentation for this class was generated from the following file:

- BootstrapSampler.hpp

13.14 BootstrapSamplerBase< Data > Class Template Reference

Base class/interface for the bootstrap sampler.

Inheritance diagram for BootstrapSamplerBase< Data >:

```
        BootstrapSamplerBase< Data >
         |                        |
         BootstrapSampler< Data >
         |                        |
          BootstrapSamplerWithGS< Data, Getter, Setter >
```
Public Member Functions

- **BootstrapSamplerBase** (size_t data_size, Data orig_data)
  
  Constructor for the bootstrap functor base.
- virtual ~BootstrapSamplerBase ()
  
  Destructor.
- virtual void operator() (size_t num_samp, Data &bootstrapped_sample) = 0
  
  Generate and store a new bootstrapped sample into bootstrapped_sample.
- virtual size_t getDataSize ()
  
  Obtain the number of samples used in the empirical distribution.
- virtual void operator() (Data &bootstrapped_sample)
  
  Generate and store a dataSize out of dataSize bootstrap sample.
- virtual Data operator() ()
  
  Return bootstrapped sample.

Static Public Member Functions

- static void set_seed (size_t seed)

Protected Attributes

- boost::random::uniform_int_distribution sampler
  
  Uniform distribution to provide samples from the empirical distribution.
- const size_t dataSize
  
  Size of the dataset defining the empirical distribution.
- Data origData
  
  Original data defining the empirical distribution TODO: Consider if it should be const (breaks Teuchos)

Static Protected Attributes

- static boost::random::mt19937 bootstrapRNG
  
  Random number generator to use for sampling.

13.14.1 Detailed Description

template<typename Data> class Dakota::BootstrapSamplerBase< Data >

Base class/interface for the bootstrap sampler.

**BootstrapSamplerBase** defines the minimum interface for a bootstrap sampler and handles initialization of the random variate generation used by the bootstrap. Functor is templated on the data type, but does not actually define a data member.
13.14.2 Member Data Documentation

boost::random::mt19937 bootstrapRNG  [static], [protected]

Random number generator to use for sampling.
    The bootstrap random number generator.
    Referenced by BootstrapSampler< Data >::operator(), BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >::operator(), and BootstrapSamplerWithGS< Data, Getter, Setter >::operator().
    The documentation for this class was generated from the following file:
    • BootstrapSampler.hpp

13.15 BootstrapSamplerWithGS< Data, Getter, Setter > Class Template Reference

A derived sampler to allow for user specification of the accessor methods.
    Inheritance diagram for BootstrapSamplerWithGS< Data, Getter, Setter >:

<table>
<thead>
<tr>
<th>BootstrapSamplerBase&lt; Data &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BootstrapSampler&lt; Data &gt;</td>
</tr>
<tr>
<td>BootstrapSamplerWithGS&lt; Data, Getter, Setter &gt;</td>
</tr>
</tbody>
</table>

Public Member Functions

• BootstrapSamplerWithGS (const Data &orig_data, Getter getter_method, Setter setter_method)
    Constructor with extra arguments for the accessor methods.
• virtual ~BootstrapSamplerWithGS ()
    Destructor.
• virtual void operator() (size_t num_samp, Data &bootstrapped_sample)
    Generate and store a new boostrapped sample into bootstrapped_sample TODO: bounds checking.

Protected Attributes

• Getter getterMethod
    Function to obtain a single sample from a Data object. Function should take a Data object and an unsigned integer corresponding to a sample index and return the sample.
• Setter setterMethod
    Function to place a single sample into a Data object. Function should take a Data object and an unsigned integer corresponding to the sample index to set.
Additional Inherited Members

13.15.1 Detailed Description

template<
typename Data, typename Getter, typename Setter>
class Dakota::BootstrapSamplerWithGS<
  Data, Getter, Setter>

A derived sampler to allow for user specification of the accessor methods.

The documentation for this class was generated from the following file:

- BootstrapSampler.hpp

13.16 callback_data Struct Reference

Public Attributes

- double rosen_cdv_upper_bd
  
  *upper bound value to pass through parser to callback function*

13.16.1 Detailed Description

Data structure to pass application-specific values through Dakota back to the callback function, for example to convey late updates to bounds, initial points, etc., to Dakota.

The documentation for this struct was generated from the following file:

- library_mode.cpp

13.17 COLINApplication Class Reference

Inherits Application< colin::MO_MINLP2::problem >.

Public Member Functions

- COLINApplication ()
  
  *Default constructor. Required by COLIN’s ApplicationHandle creation.*

- COLINApplication (Model &model)
  
  *Constructor with Model (not presently used).*

- ~COLINApplication ()
  
  *Destructor.*

- void set_problem (Model &model)
  
  *Helper function called after default construction to extract problem information from the Model and set it for COLIN.*

- void set_blocking_synch (const bool blockingSynchFlag)
  
  *Publishes whether or not COLIN is operating synchronously.*

- virtual utilib::Any spawn_evaluation_impl (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed)
  
  *Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID.*

- virtual bool evaluation_available ()
• Check to see if there are any function values ready to be collected.

• virtual void perform_evaluation_impl (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed, colin::AppResponse::response_map_t &colin_responses)
  Perform a function evaluation at t given point.

• virtual utilib::Any collect_evaluation_impl (colin::AppResponse::response_map_t &responses, utilib::seed_t &seed)
  Collect a completed evaluation from DAKOTA.

• virtual void colin_request_to_dakota_request (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed)
  Helper function to convert evaluation request data from COLIN structures to DAKOTA structures.

• virtual void dakota_response_to_colin_response (const Response &dakota_response, colin::AppResponse::response_map_t &colin_responses)
  Helper function to convert evaluation response data from DAKOTA structures to COLIN structures.

• virtual bool map_domain (const utilib::Any &src, utilib::Any &native, bool forward=true) const
  Map the domain point into data type desired by this application context.

Protected Attributes

• Model iteratedModel
  Shallow copy of the model on which COLIN will iterate.

• bool blockingSynch
  Flag for COLIN synchronous behavior (Pattern Search only).

• ActiveSet activeSet
  Local copy of model’s active set for convenience.

• std::vector<int> requestedEvals
  Evaluations queued for async evaluation.

• IntResponseMap dakota_responses
  eval_id to response mapping to cache completed jobs.

13.17.1 Detailed Description

COLINAApplication is a DAKOTA class that is derived from COLIN’s Application hierarchy. It redefines a variety of virtual COLIN functions to use the corresponding DAKOTA functions. This is a more flexible algorithm library interfacing approach than can be obtained with the function pointer approaches used by NPSOLOptimizer and SNLLOptimizer.

13.17.2 Member Function Documentation

void set_problem ( Model & model )

Helper function called after default construction to extract problem information from the Model and set it for COLIN.

Set variable bounds and linear and nonlinear constraints. This avoids using probDescDB, so it is called by both the standard and the on-the-fly COLINOptimizier constructors.

References Response::active_set(), COLINAApplication::activeSet, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Dakota::copy_data_partial(), Model::current_response(), Model::cv(), Model::discrete_int_lower_bounds(), Model::discrete_int_sets(), Model::discrete_int_upper_bounds(), Model::discrete_set_int_values(),
Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::div(), Model::drv(), Model::dsv(),
Dakota::get_nonlinear_constraints(), COLINApplication::iteratedModel, Model::linear_eq_constraint_coeffs(), Model-
::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(),
Model::linear_ineq_constraint_upper_bounds(), Model::num_functions(), Model::num_linear_eq_constraints(), Model-
::num_linear_ineq_constraints(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(),
and Model::primary_response_fn_sense().

Referenced by COLINApplication::COLINApplication().

utilib::Any spawn_evaluation_impl ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests,
utilib::seed_t & seed ) [virtual]

Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID.
Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID. This is only called by COLIN’s concurrent evaluator, which is only instantiated when the Model supports asynch evals. The domain point is guaranteed to be compatible with data type specified by map_domain(...)
References COLINApplication::colin_request_to_dakota_request(), Model::evaluate_nowait(), Model::evaluation_id(), and COLINApplication::iteratedModel.

bool evaluation_available ( ) [virtual]

Check to see if there are any function values ready to be collected.
Check to see if any asynchronous evaluations have finished. This is only called by COLIN’s concurrent evaluator, which is only instantiated when the Model supports asynch evals.
References COLINApplication::blockingSynch, COLINApplication::dakota_responses, COLINApplication::iteratedModel, Model::synchronize(), and Model::synchronize_nowait().

void perform_evaluation_impl ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests,
utilib::seed_t & seed, colin::AppResponse::response_map_t & colin_responses ) [virtual]

Perform a function evaluation at t given point.
Perform an evaluation at a specified domain point. Wait for and return the response. This is only called by COLIN’s serial evaluator, which is only instantiated when the Model does not support asynch evals. The domain point is guaranteed to be compatible with data type specified by map_domain(...)
References COLINApplication::colin_request_to_dakota_request(), Model::current_response(), COLINApplication::dakota_response_to_colin_response(), Model::evaluate(), and COLINApplication::iteratedModel.

utilib::Any collect_evaluation_impl ( colin::AppResponse::response_map_t & colin_responses,
utilib::seed_t & seed ) [virtual]

Collect a completed evaluation from DAKOTA.
Collect the next completed evaluation from DAKOTA. Always returns the evalid of the response returned.
References COLINApplication::dakota_response_to_colin_response(), and COLINApplication::dakota_responses.

void colin_request_to_dakota_request ( const utilib::Any & domain, const
colin::AppRequest::request_map_t & requests, utilib::seed_t & seed ) [virtual]

Helper function to convert evaluation request data from COLIN structures to DAKOTA structures.
Map COLIN info requests to DAKOTA objectives and constraints.
References Model::continuous_variables(), Model::discrete_int_sets(), Model::discrete_int_variable(), Model::discrete_real_variable(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::discrete_string_variable(), Model::div(), Model::drv(), Model::dsv(), COLINApplication::iteratedModel, Model::num_functions(), and Dakota::set_index_to_value().

Referenced by COLINApplication::perform_evaluation_impl(), and COLINApplication::spawn_evaluation_impl().

```cpp
void dakota_response_to_colin_response ( const Response & dakota_response,
        colin::AppResponse::response_map_t & colin_responses ) [virtual]
```

Gelper function to convert evaluation response data from DAKOTA structures to COLIN structures.

Map DAKOTA objective and constraint values to COLIN response.

References Response::active_set_request_vector(), and Response::function_value().

Referenced by COLINApplication::collect_evaluation_impl(), and COLINApplication::perform_evaluation_impl().

```cpp
bool map_domain ( const utilib::Any & src, utilib::Any & native, bool forward = true ) const [virtual]
```

Map the domain point into data type desired by this application context.

Map the domain point into data type desired by this application context (utilib::MixedIntVars). This data type can be exposed from the Any &domain presented to spawn and collect.

The documentation for this class was generated from the following files:

- COLINApplication.hpp
- COLINApplication.cpp

### 13.18 COLINTraits Class Reference

Wrapper class for optimizers defined using COLIN.

Inheritance diagram for COLINTraits:

```
TraitsBase

↓

COLINTraits
```

**Public Member Functions**

- **COLINTraits ()**
  
  *default constructor*

- **virtual ~COLINTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*
13.19. COLLABHYBRIDMETAITERATOR CLASS REFERENCE

- bool supports_nonlinear_equality ()
  Return the flag indicating whether method supports nonlinear equalities.
- bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.
- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  Return the format used for nonlinear inequality constraints.

13.18.1 Detailed Description

Wrapper class for optimizers defined using COLIN.

The COLINOptimizer class wraps COLIN, a Sandia-developed C++ optimization interface library. A variety of COLIN optimizers are defined in COLIN and its associated libraries, including SCOLIB which contains the optimization components from the old COLINY (formerly SGOPT) library. COLIN contains optimizers such as genetic algorithms, pattern search methods, and other nongradient-based techniques. COLINOptimizer uses a COLINApplication object to perform the function evaluations.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, and solution_accuracy are mapped into COLIN’s max_iterations, max_function_evaluations, this_trial, function_value_tolerance, sufficient_objective_value properties. An outputLevel is mapped to COLIN’s output_level property and a setting of debug activates output of method initialization and sets the COLIN debug attribute to 10000 for the DEBUG output level. Refer to [Hart, W.E., 2006] for additional information on COLIN objects and controls. A version of TraitsBase specialized for COLIN optimizers.

The documentation for this class was generated from the following file:

- COLINOptimizer.hpp

13.19 CollabHybridMetaIterator Class Reference

Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.

Inheritance diagram for CollabHybridMetaIterator:

```
CollabHybridMetaIterator
  MetaIterator
    Iterator
    CollabHybridMetaIterator
```

Public Member Functions

- CollabHybridMetaIterator (ProblemDescDB &problem_db)
  standard constructor
- CollabHybridMetaIterator (ProblemDescDB &problem_db, Model &model)
  alternate constructor
- ~CollabHybridMetaIterator ()
  destructor
Protected Member Functions

- void core_run ()
  
  Performs the collaborative hybrid iteration.

- void derived_init_communicators (ParLevLIter pl_iter)
  
  derived class contributions to initializing the communicators associated with this Iterator instance

- void derived_set_communicators (ParLevLIter pl_iter)
  
  derived class contributions to setting the communicators associated with this Iterator instance

- void derived_free_communicators (ParLevLIter pl_iter)
  
  derived class contributions to freeing the communicators associated with this Iterator instance

- IntIntPair estimate_partition_bounds ()
  
  estimate the minimum and maximum partition sizes that can be utilized by this Iterator

- const Variables & variables_results () const
  
  return the final solution from the collaborative iteration (variables)

- const Response & response_results () const
  
  return the final solution from the collaborative iteration (response)

Private Attributes

- String hybridCollabType
  
  abo or hops

- StringArray methodStrings
  
  the list of method pointer or method name identifiers

- StringArray modelStrings
  
  the list of model pointer identifiers for method identification by name

- bool lightwtMethodCtor
  
  use of lightweight Iterator construction by name

- bool singlePassedModel
  
  use of constructor that enforces use of a single passed Model

- IteratorArray selectedIterators
  
  the set of iterators, one for each entry in methodStrings

- ModelArray selectedModels
  
  the set of models, one for each iterator

- Variables bestVariables
  
  best variables found in collaborative iteration

- Response bestResponse
  
  best response found in collaborative iteration

Additional Inherited Members

13.19.1 Detailed Description

Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.

This meta-iterator has two approaches to hybrid iteration: (1) agent-based using the ABO framework; (2) nonagent-based using the HOPSPACK framework.

The documentation for this class was generated from the following files:
13.20. **CommandLineHandler Class Reference**

Utility class for managing command line inputs to DAKOTA.

Inheritance diagram for CommandLineHandler:

```
CommandLineHandler
    └── GetLongOpt
        └── CommandLineHandler
```

### Public Member Functions

- **CommandLineHandler ()**
  
  Default constructor, requires `check_usage()` call for parsing

- **CommandLineHandler (int argc, char **argv, int world_rank)**
  
  Constructor with parsing

- **~CommandLineHandler ()**
  
  Destructor

- **void check_usage (int argc, char **argv)**
  
  Verifies that DAKOTA is called with the correct command usage. Prints a descriptive message and exits the program if incorrect.

- **int read_restart_evals () const**
  
  Returns the number of evaluations to be read from the restart file (as specified on the DAKOTA command line) as an integer instead of a const char*.

- **void usage (std::ostream &outfile=Cout) const**
  
  Prints usage information to outfile, conditionally on rank.

### Private Member Functions

- **void initialize_options ()**
  
  Enrolls the supported command line inputs.

- **void output_helper (const std::string &message, std::ostream &os) const**
  
  Output only on Dakota worldRank 0 if possible

### Private Attributes

- **int worldRank**
  
  Rank of this process within Dakota's allocation; manages conditional output.
Additional Inherited Members

13.20.1 Detailed Description

Utility class for managing command line inputs to DAKOTA.

**CommandLineHandler** provides additional functionality that is specific to DAKOTA’s needs for the definition and parsing of command line options. Inheritance is used to allow the class to have all the functionality of the base class, GetLongOpt.

13.20.2 Member Function Documentation

```cpp
void output_helper(const std::string &message, std::ostream &os) const [private]
```

output only on Dakota worldRank 0 if possible

When there is a valid ParallelLibrary, output only on rank 0

References CommandLineHandler::worldRank.

Referenced by CommandLineHandler::check_usage().

The documentation for this class was generated from the following files:

- CommandLineHandler.hpp
- CommandLineHandler.cpp

13.21 CommandShell Class Reference

Utility class which defines convenience operators for spawning processes with system calls.

Public Member Functions

- **CommandLineShell ()**
  constructor
- **~CommandLineShell ()**
  destructor
- **CommandLineShell & operator<< (const char *cmd)**
  appends cmd to sysCommand
- **CommandLineShell & operator<< (const std::string &cmd)**
  convenient operator: appends string to the commandString to be executed
- **CommandLineShell & operator<< (CommandLineShell &(*f)(CommandLineShell &))**
  allows passing of the flush function to the shell using <<
- **CommandLineShell & flush ()**
  "flushes" the shell; i.e. executes the sysCommand
- **void asynch_flag (const bool flag)**
  set the asynchFlag
- **bool asynch_flag () const**
  get the asynchFlag
- **void suppress_output_flag (const bool flag)**
  set the suppressOutputFlag
- **bool suppress_output_flag () const**
  get the suppressOutputFlag
Private Attributes

- std::string sysCommand
  
  *The command string that is constructed through one or more << insertions and then executed by flush.*

- bool asynchFlag
  
  *flags nonblocking operation (background system calls)*

- bool suppressOutputFlag
  
  *flags suppression of shell output (no command echo)*

13.21.1 Detailed Description

Utility class which defines convenience operators for spawning processes with system calls.

The CommandShell class wraps the C system() utility and defines convenience operators for building a command string and then passing it to the shell.

13.21.2 Member Function Documentation

CommandShell & operator<< ( const char * cmd ) [inline]

appends cmd to sysCommand

  *convenient operator: appends string to the commandString to be executed*

  References CommandShell::sysCommand.

CommandShell & operator<< ( CommandShell &(*)(CommandShell &) f ) [inline]

allows passing of the flush function to the shell using <<

  *convenience operator: allows passing of the flush func to the shell via <<*

CommandShell & flush()

"flushes" the shell; i.e. executes the sysCommand

  Executes the sysCommand by passing it to system(). Appends an "&" if asynchFlag is set (background system call) and echos the sysCommand to Cout if suppressOutputFlag is not set.

  References Dakota::abort_handler(), CommandShell::asynchFlag, CommandShell::suppressOutputFlag, and CommandShell::sysCommand.

  Referenced by Dakota::flush().

The documentation for this class was generated from the following files:

- CommandShell.hpp
- CommandShell.cpp

13.22 ConcurrentMetaIterator Class Reference

Meta-iterator for multi-start iteration or pareto set optimization.

Inheritance diagram for ConcurrentMetaIterator:
CHAPTER 13. CLASS DOCUMENTATION

```
  Iterator
   | 
  MetaIterator
   | 
ConcurrentMetaIterator

Public Member Functions

- ConcurrentMetaIterator (ProblemDescDB &problem_db)
  standard constructor
- ConcurrentMetaIterator (ProblemDescDB &problem_db, Model &model)
  alternate constructor
- ~ConcurrentMetaIterator ()
  destructor

Protected Member Functions

- void pre_run()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void core_run()
  Performs the concurrent iteration by executing selectedIterator on iteratedModel multiple times in parallel for different parameter sets.
- void print_results(std::ostream &s, short results_state=FINALRESULTS)
  print the final iterator results
- void derived_init_communicators(ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators(ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators(ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- IntIntPair estimate_partition_bounds()
  estimate the minimum and maximum partition sizes that can be utilized by this Iterator
- void initialize_iterator(int job_index)
  used by IteratorScheduler to set the starting data for a run
- void pack_parameters_buffer(MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack starting data for an iterator run
- void unpack_parameters_initialize(MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack starting data and initialize an iterator run
- void pack_results_buffer(MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack results data from an iterator run
- void unpack_results_buffer(MPIUnpackBuffer &recv_buffer, int job_index)
```
used by IteratorScheduler to unpack results data from an iterator run

- void update_local_results (int job_index)
  used by IteratorScheduler to update local results arrays

- const Model & algorithm_space_model () const
  return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain

Private Member Functions

- void initialize_iterator (const RealVector &param_set)
  called by unpack_parameters_initialize(MPIUnpackBuffer) and initialize_iterator(int) to update iteratedModel and selectedIterator

- void initialize_model ()
  initialize the iterated Model prior to Iterator instantiation and define param_set_len

Private Attributes

- Iterator selectedIterator
  the iterator selected for concurrent iteration

- RealVector initialPt
  the initial continuous variables for restoring the starting point in the Pareto set minimization

- RealVectorArray parameterSets
  an array of parameter set vectors (either multistart variable sets or pareto multi-objective/least squares weighting sets) to be performed.

- int paramSetLen
  length of each of the parameter sets associated with an iterator job (number of continuous variables for MULTI_START, number of objective fns for PARETO_SET)

- int numRandomJobs
  number of randomly-generated parameter sets to evaluate

- int randomSeed
  seed for random number generator for random samples

- PRPArray prpResults
  1-d array of ParamResponsePair results corresponding to numIteratorJobs

Friends

- class IteratorScheduler
  protect scheduler callback functions from general access

Additional Inherited Members

13.22.1 Detailed Description

Meta-iterator for multi-start iteration or pareto set optimization.

This meta-iterator maintains two concurrent iterator capabilities. First, a general capability for running an iterator multiple times from different starting points is provided (often used for multi-start optimization, but not restricted to optimization). Second, a simple capability for mapping the "pareto frontier" (the set of optimal
CHAPTER 13. CLASS DOCUMENTATION

solutions in multiobjective formulations) is provided. This pareto set is mapped through running an optimizer multiple times for different sets of multiobjective weightings.

13.22.2 Member Function Documentation

void pre_run( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Iterator.

References Analyzer::all_samples(), Iterator::all_samples(), ParallelLibrary::bcast hs(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Dakota::copy_data(), Model::estimate_message_lengths(), ConcurrentMetaIterator::initialPt, Iterator::iteratedModel, IteratorScheduler::iterator_scheduling(), IteratorScheduler::iteratorServerId, MetaIterator::iterSched, IteratorScheduler::lead_rank(), Model::message_lengths(), Iterator::method_Name, Iterator::methodPCIter, IteratorScheduler::miPLIndex, IteratorScheduler::numIteratorJobs, IteratorScheduler::numIteratorServers, ConcurrentMetaIterator::numRandomJobs, Iterator::parallelLib, ConcurrentMetaIterator::parameterSets, ConcurrentMetaIterator::paramSetLen, ConcurrentMetaIterator::prpResults, ConcurrentMetaIterator::randomSeed, and MPIPackBuffer::size().

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Iterator.

References Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), ParamResponsePair::eval_id(), Iterator::methodName, ConcurrentMetaIterator::parameterSets, ConcurrentMetaIterator::paramSetLen, ConcurrentMetaIterator::prpResults, ParamResponsePair::response(), ParamResponsePair::variables(), and Response::write_tabular().

The documentation for this class was generated from the following files:

- ConcurrentMetaIterator.hpp
- ConcurrentMetaIterator.cpp

13.23 CONMINTraits Class Reference

Wrapper class for the CONMIN optimization library.

Inheritance diagram for CONMINTraits:

```
TraitsBase
  \|-- CONMINTraits
```
Public Member Functions

- CONMINTraits ()
  default constructor
- virtual ~CONMINTraits ()
  destructor
- virtual bool is_derived ()
  A temporary query used in the refactor.
- bool supports_continuous_variables ()
  Return the flag indicating whether method supports continuous variables.
- bool supports_linear_equality ()
  Return the flag indicating whether method supports linear equalities.
- bool supports_linear_inequality ()
  Return the flag indicating whether method supports linear inequalities.
- bool supports_nonlinear_equality ()
  Return the flag indicating whether method supports nonlinear equalities.
- bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.
- LINEAR_INEQUALITY_FORMAT linear_inequality_format ()
  Return the format used for linear inequality constraints.
- bool supports_nonlinear_equality ()
  Return the flag indicating whether method supports nonlinear equalities.
- bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.
- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  Return the format used for nonlinear inequality constraints.

13.23.1 Detailed Description

Wrapper class for the CONMIN optimization library.

The CONMINOptimizer class provides a wrapper for CONMIN, a Public-domain Fortran 77 optimization library written by Gary Vanderplaats under contract to NASA Ames Research Center. The CONMIN User's Manual is contained in NASA Technical Memorandum X-62282, 1978. CONMIN uses a reverse communication mode, which avoids the static member function issues that arise with function pointer designs (see NPSOL-Optimizer and SNLLOptimizer).

The user input mappings are as follows: max.iterations is mapped into CONMIN's ITMAX parameter, max.function_evaluations is implemented directly in the core.run() loop since there is no CONMIN parameter equivalent, convergence_tolerance is mapped into CONMIN's DELFUN and DABFUN parameters, output verbosity is mapped into CONMIN's IPRINT parameter (verbose: IPRINT = 4; quiet: IPRINT = 2), gradient mode is mapped into CONMIN's NFDF parameter, and finite difference step size is mapped into CONMIN's FDCH and FDCHM parameters. Refer to [Vanderplaats, 1978] for additional information on CONMIN parameters. A version of TraitsBase specialized for CONMIN optimizers

The documentation for this class was generated from the following file:

- CONMINOptimizer.hpp

13.24 ConsoleRedirector Class Reference

Public Member Functions

- ConsoleRedirector (std::ostream *dakota_stream, std::ostream *default_dest)
Constructor taking a reference to the Dakota COUT/CERR handle and a default destination to use when no redirection (or destruct)

- \(~\text{ConsoleRedirector}()\)
  when the redirector stack is destroyed, it will rebind the output handle to the default ostream, then destroy open files

- \text{void push\_back()}\)
  push back the default or repeat the last pushed file stream

- \text{void push\_back(const String &filename)}\)
  push back a new output filestream, or repeat the last one if no filename change

- \text{void pop\_back()}\)
  pop the last redirection

Protected Attributes

- \text{std::ostream *\& ostreamHandle}\)
  The handle (target ostream) through which output is sent; typically dakota\_cout or dakota\_cerr. Will be rebound to specific streams as they are pushed or popped.

- \text{std::ostream *defaultOStream}\)
  initial stream to reset to when redirections are done (typically std::cout or std::cerr)

- \text{std::vector< boost::shared\_ptr < OutputWriter > > ostreamDestinations}\)
  stack of redirections to OutputWriters; shared pointers are used to potentially share the same ostream at multiple levels

Private Member Functions

- \text{ConsoleRedirector()}\)
  default constructor is disallowed

- \text{ConsoleRedirector(const ConsoleRedirector \&)}\)
  copy constructor is disallowed due

- \text{const ConsoleRedirector \& operator= (const ConsoleRedirector \&)}\)
  assignment is disallowed

13.24.1 Detailed Description

Component to manage a set of output or error redirections. Push operations may present a new filename, or none in order to preserve current binding to cout/cerr or file, but place an entry on the stack. Cout/Cerr are rebound as needed when a stream is destroyed on pop.

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp
13.25 Constraints Class Reference

Base class for the variable constraints class hierarchy.

Inheritance diagram for Constraints:

```
  Constraints
     |     |
     v     v
 MixedVarConstraints  RelaxedVarConstraints
```

Public Member Functions

- Constraints ()
  default constructor
- Constraints (const ProblemDescDB &prob_db, const SharedVariablesData &svd)
  standard constructor
- Constraints (const SharedVariablesData &svd)
  alternate constructor for instantiations on the fly
- Constraints (const Constraints &con)
  copy constructor
- virtual ~Constraints ()
  destructor
- Constraints operator= (const Constraints &con)
  assignment operator
- virtual void write (std::ostream &s) const
  write a variable constraints object to an std::ostream
- virtual void read (std::istream &s)
  read a variable constraints object from an std::istream
- const RealVector & continuous_lower_bounds () const
  return the active continuous variable lower bounds
- Real continuous_lower_bound (size_t i) const
  return an active continuous variable lower bound
- void continuous_lower_bounds (const RealVector &cl_bnds)
  set the active continuous variable lower bounds
- void continuous_lower_bound (Real cl_bnd, size_t i)
  set an active continuous variable lower bound
- const RealVector & continuous_upper_bounds () const
  return the active continuous variable upper bounds
- Real continuous_upper_bound (size_t i) const
  return an active continuous variable upper bound
- void continuous_upper_bounds (const RealVector &cu_bnds)
  set the active continuous variable upper bounds
- void continuous_upper_bound (Real cu_bnd, size_t i)
set an active continuous variable upper bound
- const IntVector & \texttt{discrete\_int\_lower\_bounds} () const
  return the active discrete variable lower bounds
- int \texttt{discrete\_int\_lower\_bound} (size\_t i) const
  return an active discrete variable lower bound
- void \texttt{discrete\_int\_lower\_bounds} (const IntVector &dil\_bnds)
  set the active discrete variable lower bounds
- void \texttt{discrete\_int\_lower\_bound} (int dil\_bnd, size\_t i)
  set an active discrete variable lower bound
- const IntVector & \texttt{discrete\_int\_upper\_bounds} () const
  return the active discrete variable upper bounds
- int \texttt{discrete\_int\_upper\_bound} (size\_t i) const
  return an active discrete variable upper bound
- void \texttt{discrete\_int\_upper\_bounds} (const IntVector &diu\_bnds)
  set the active discrete variable upper bounds
- void \texttt{discrete\_int\_upper\_bound} (int diu\_bnd, size\_t i)
  set an active discrete variable upper bound
- const RealVector & \texttt{discrete\_real\_lower\_bounds} () const
  return the active discrete variable lower bounds
- Real \texttt{discrete\_real\_lower\_bound} (size\_t i) const
  return an active discrete variable lower bound
- void \texttt{discrete\_real\_lower\_bounds} (const RealVector &drl\_bnds)
  set the active discrete variable lower bounds
- void \texttt{discrete\_real\_lower\_bound} (Real drl\_bnd, size\_t i)
  set an active discrete variable lower bound
- const RealVector & \texttt{discrete\_real\_upper\_bounds} () const
  return the active discrete variable upper bounds
- Real \texttt{discrete\_real\_upper\_bound} (size\_t i) const
  return an active discrete variable upper bound
- void \texttt{discrete\_real\_upper\_bounds} (const RealVector &dru\_bnds)
  set the active discrete variable upper bounds
- void \texttt{discrete\_real\_upper\_bound} (Real dru\_bnd, size\_t i)
  set an active discrete variable upper bound
- const RealVector & \texttt{inactive\_continuous\_lower\_bounds} () const
  return the inactive continuous lower bounds
- void \texttt{inactive\_continuous\_lower\_bounds} (const RealVector &icl\_bnds)
  set the inactive continuous lower bounds
- const RealVector & \texttt{inactive\_continuous\_upper\_bounds} () const
  return the inactive continuous upper bounds
- void \texttt{inactive\_continuous\_upper\_bounds} (const RealVector &icu\_bnds)
  set the inactive continuous upper bounds
- const IntVector & \texttt{inactive\_discrete\_int\_lower\_bounds} () const
  return the inactive discrete lower bounds
- void \texttt{inactive\_discrete\_int\_lower\_bounds} (const IntVector &icl\_bnds)
13.25. CONSTRAINTS CLASS REFERENCE

- void inactive_discrete_int_lower_bounds (const IntVector &idil_bnds)
  set the inactive discrete lower bounds
- const IntVector & inactive_discrete_int_upper_bounds () const
  return the inactive discrete upper bounds
- void inactive_discrete_int_upper_bounds (const IntVector &idiu_bnds)
  set the inactive discrete upper bounds
- const RealVector & inactive_discrete_real_lower_bounds () const
  return the inactive discrete lower bounds
- void inactive_discrete_real_lower_bounds (const RealVector &idrl_bnds)
  set the inactive discrete lower bounds
- const RealVector & inactive_discrete_real_upper_bounds () const
  return the inactive discrete upper bounds
- void inactive_discrete_real_upper_bounds (const RealVector &idru_bnds)
  set the inactive discrete upper bounds
- const RealVector & all_continuous_lower_bounds () const
  returns a single array with all continuous lower bounds
- void all_continuous_lower_bounds (const RealVector &acl_bnds)
  sets all continuous lower bounds using a single array
- void all_continuous_lower_bound (Real acl_bnd, size_t i)
  set a lower bound within the all continuous lower bounds array
- const RealVector & all_continuous_upper_bounds () const
  returns a single array with all continuous upper bounds
- void all_continuous_upper_bounds (const RealVector &acu_bnds)
  sets all continuous upper bounds using a single array
- void all_continuous_upper_bound (Real acu_bnd, size_t i)
  set an upper bound within the all continuous upper bounds array
- const IntVector & all_discrete_int_lower_bounds () const
  returns a single array with all discrete lower bounds
- void all_discrete_int_lower_bounds (const IntVector &adil_bnds)
  sets all discrete lower bounds using a single array
- void all_discrete_int_lower_bound (int adil_bnd, size_t i)
  set a lower bound within the all discrete lower bounds array
- const IntVector & all_discrete_int_upper_bounds () const
  returns a single array with all discrete upper bounds
- void all_discrete_int_upper_bounds (const IntVector &adiu_bnds)
  sets all discrete upper bounds using a single array
- void all_discrete_int_upper_bound (int adiu_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array
- const RealVector & all_discrete_real_lower_bounds () const
  returns a single array with all discrete lower bounds
- void all_discrete_real_lower_bounds (const RealVector &adrl_bnds)
  sets all discrete lower bounds using a single array
- void all_discrete_real_lower_bound (Real adrl_bnd, size_t i)
  set a lower bound within the all discrete lower bounds array
- const RealVector & all_discrete_real_upper_bounds () const
  returns a single array with all discrete upper bounds
- void all_discrete_real_upper_bounds (const RealVector &adru_bnds)
  sets all discrete upper bounds using a single array
- void all_discrete_real_upper_bound (Real adru_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array
set a lower bound within the all discrete lower bounds array
• const RealVector & all_discrete_real_lower_bounds() const
  returns a single array with all discrete lower bounds
• void all_discrete_real_lower_bounds(const RealVector &adru_bnds)
  sets all discrete lower bounds using a single array
• void all_discrete_real_upper_bound(Real adru_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array
• size_t num_linear_ineq_constraints() const
  return the number of linear inequality constraints
• size_t num_linear_eq_constraints() const
  return the number of linear equality constraints
• const RealMatrix & linear_ineq_constraint_coeffs() const
  return the linear inequality constraint coefficients
• void linear_ineq_constraint_coeffs(const RealMatrix &lin_ineq_coeffs)
  set the linear inequality constraint coefficients
• const RealVector & linear_ineq_constraint_lower_bounds() const
  return the linear inequality constraint lower bounds
• void linear_ineq_constraint_lower_bounds(const RealVector &lin_ineq_l_bnds)
  set the linear inequality constraint lower bounds
• const RealVector & linear_ineq_constraint_upper_bounds() const
  return the linear inequality constraint upper bounds
• void linear_ineq_constraint_upper_bounds(const RealVector &lin_ineq_u_bnds)
  set the linear inequality constraint upper bounds
• const RealMatrix & linear_eq_constraint_coeffs() const
  return the linear equality constraint coefficients
• void linear_eq_constraint_coeffs(const RealMatrix &lin_eq_coeffs)
  set the linear equality constraint coefficients
• const RealVector & linear_eq_constraint_targets() const
  return the linear equality constraint targets
• void linear_eq_constraint_targets(const RealVector &lin_eq_targets)
  set the linear equality constraint targets
• size_t num_nonlinear_ineq_constraints() const
  return the number of nonlinear inequality constraints
• size_t num_nonlinear_eq_constraints() const
  return the number of nonlinear equality constraints
• const RealVector & nonlinear_ineq_constraint_lower_bounds() const
  return the nonlinear inequality constraint lower bounds
• void nonlinear_ineq_constraint_lower_bounds(const RealVector &nln_ineq_l_bnds)
  set the nonlinear inequality constraint lower bounds
• const RealVector & nonlinear_ineq_constraint_upper_bounds() const
  return the nonlinear inequality constraint upper bounds
• void nonlinear_ineq_constraint_upper_bounds(const RealVector &nln_ineq_u_bnds)
  set the nonlinear inequality constraint upper bounds
• const RealVector & nonlinear_eq_constraint_targets () const
  return the nonlinear equality constraint targets
• void nonlinear_eq_constraint_targets (const RealVector &nln_eq_targets)
  set the nonlinear equality constraint targets
• Constraints copy () const
  for use when a deep copy is needed (the representation is not shared)
• void shape ()
  shape the lower/upper bound arrays based on sharedVarsData
• void reshape (size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons, const SharedVariablesData &svd)
  reshape the linear/nonlinear/bound constraint arrays arrays and the lower/upper bound arrays
• void reshape ()
  reshape the lower/upper bound arrays based on sharedVarsData
• void reshape (size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons)
  reshape the linear/nonlinear constraint arrays
• void inactive_view (short view2)
  sets the inactive view based on higher level (nested) context
• bool is_null () const
  function to check constraintsRep (does this envelope contain a letter)

Protected Member Functions

• Constraints (BaseConstructor, const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
• Constraints (BaseConstructor, const SharedVariablesData &svd)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
• void build_views ()
  construct active/inactive views of all variables arrays
• void build_active_views ()
  construct active views of all variables bounds arrays
• void build_inactive_views ()
  construct inactive views of all variables bounds arrays
• void manage_linear_constraints (const ProblemDescDB &problem_db)
  perform checks on user input, convert linear constraint coefficient input to matrices, and assign defaults

Protected Attributes

• SharedVariablesData sharedVarsData
  configuration data shared from a Variables instance
• RealVector allContinuousLowerBnds
  a continuous lower bounds array combining continuous design, uncertain, and continuous state variable types (all view).
• RealVector allContinuousUpperBnds
  a continuous upper bounds array combining continuous design, uncertain, and continuous state variable types (all view).
• IntVector allDiscreteIntLowerBnds
  a discrete lower bounds array combining discrete design and discrete state variable types (all view).
• IntVector allDiscreteIntUpperBnds
  a discrete upper bounds array combining discrete design and discrete state variable types (all view).
• RealVector allDiscreteRealLowerBnds
  a discrete lower bounds array combining discrete design and discrete state variable types (all view).
• RealVector allDiscreteRealUpperBnds
  a discrete upper bounds array combining discrete design and discrete state variable types (all view).
• size_t numNonlinearIneqCons
  number of nonlinear inequality constraints
• size_t numNonlinearEqCons
  number of nonlinear equality constraints
• RealVector nonlinearIneqConLowerBnds
  nonlinear inequality constraint lower bounds
• RealVector nonlinearIneqConUpperBnds
  nonlinear inequality constraint upper bounds
• RealVector nonlinearEqConTargets
  nonlinear equality constraint targets
• size_t numLinearIneqCons
  number of linear inequality constraints
• size_t numLinearEqCons
  number of linear equality constraints
• RealMatrix linearIneqConCoeffs
  linear inequality constraint coefficients
• RealMatrix linearEqConCoeffs
  linear equality constraint coefficients
• RealVector linearIneqConLowerBnds
  linear inequality constraint lower bounds
• RealVector linearIneqConUpperBnds
  linear inequality constraint upper bounds
• RealVector linearEqConTargets
  linear equality constraint targets
• RealVector continuousLowerBnds
  the active continuous lower bounds array view
• RealVector continuousUpperBnds
  the active continuous upper bounds array view
• IntVector discreteIntLowerBnds
  the active discrete lower bounds array view
• IntVector discreteIntUpperBnds
  the active discrete upper bounds array view
13.25. CONSTRAINTS CLASS REFERENCE

- RealVector `discreteRealLowerBnds`
  
  the active discrete lower bounds array view

- RealVector `discreteRealUpperBnds`
  
  the active discrete upper bounds array view

- RealVector `inactiveContinuousLowerBnds`
  
  the inactive continuous lower bounds array view

- RealVector `inactiveContinuousUpperBnds`
  
  the inactive continuous upper bounds array view

- IntVector `inactiveDiscreteIntLowerBnds`
  
  the inactive discrete lower bounds array view

- IntVector `inactiveDiscreteIntUpperBnds`
  
  the inactive discrete upper bounds array view

- RealVector `inactiveDiscreteRealLowerBnds`
  
  the inactive discrete lower bounds array view

- RealVector `inactiveDiscreteRealUpperBnds`
  
  the inactive discrete upper bounds array view

### Private Member Functions

- Constraints * `get_constraints` (const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  
  Used only by the constructor to initialize constraintsRep to the appropriate derived type.

- Constraints * `get_constraints` (const SharedVariablesData &svd) const
  
  Used by copy() to initialize constraintsRep to the appropriate derived type.

### Private Attributes

- Constraints * `constraintsRep`
  
  pointer to the letter (initialized only for the envelope)

- int `referenceCount`
  
  number of objects sharing constraintsRep

13.25.1 Detailed Description

Base class for the variable constraints class hierarchy.

The Constraints class is the base class for the class hierarchy managing bound, linear, and nonlinear constraints. Using the variable lower and upper bounds arrays from the input specification, different derived classes define different views of this data. The linear and nonlinear constraint data is consistent in all views and is managed at the base class level. For memory efficiency and enhanced polymorphism, the variable constraints hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Constraints) serves as the envelope and one of the derived classes (selected in Constraints::get_constraints()) serves as the letter.
13.25.2 Constructor & Destructor Documentation

Constraints ( )

default constructor

The default constructor: constraintsRep is NULL in this case (a populated problem db is needed to build a meaningful Constraints object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

Constraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )

standard constructor

The envelope constructor only needs to extract enough data to properly execute get_constraints, since the constructor overloaded with BaseConstructor builds the actual base class data inherited by the derived classes.

References Dakota::abort_handler(), Constraints::constraintsRep, and Constraints::get_constraints().

Constraints ( const SharedVariablesData & svd )

alternate constructor for instantiations on the fly

Envelope constructor for instantiations on the fly. This constructor executes get_constraints(view), which invokes the default derived/base constructors, followed by a reshape() based on vars_comps.

References Dakota::abort_handler(), Constraints::constraintsRep, and Constraints::get_constraints().

Constraints ( const Constraints & con )

copy constructor

Copy constructor manages sharing of constraintsRep and incrementing of referenceCount.

References Constraints::constraintsRep, and Constraints::referenceCount.

~Constraints ( ) [virtual]

destructor

Destructor decrements referenceCount and only deletes constraintsRep when referenceCount reaches zero.

References Constraints::constraintsRep, and Constraints::referenceCount.

Constraints ( BaseConstructor, const ProblemDescDB & problem_db, const SharedVariablesData & svd ) [protected]

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_constraints() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_constraints() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Constraints).

References Constraints::build_views(), Constraints::manage_linear_constraints(), and Constraints::shape().

Constraints ( BaseConstructor, const SharedVariablesData & svd ) [protected]

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_constraints() instantiate a derived class letter and the derived constructor selects this base class constructor in its initialization list (to
avoid recursion in the base class constructor calling get_constraints() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Constraints).

References Constraints::build_views(), and Constraints::shape().

### 13.25.3 Member Function Documentation

**Constraints operator= ( const Constraints & con )**

assignment operator


  References Constraints::constraintsRep, and Constraints::referenceCount.

**Constraints copy ( ) const**

for use when a deep copy is needed (the representation is not shared)

Deep copies are used for history mechanisms that catalogue permanent copies (should not change as the representation within userDefinedConstraints changes).

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Constraints::build_views(), Constraints::constraintsRep, Constraints::get_constraints(), Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearIneqConCoeffs, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::nonlinearEqConTargets, Constraints::nonlinearIneqConLowerBnds, Constraints::nonlinearIneqConUpperBnds, Constraints::numLinearEqCons, Constraints::numNonlinearEqCons, Constraints::numNonlinearIneqCons, and Constraints::sharedVarsData.

Referenced by SurrogateModel::force_rebuild(), and RecastModel::init_constraints().

**void shape ( )**

shape the lower/upper bound arrays based on sharedVarsData

- Resizes the derived bounds arrays.

References SharedVariablesData::all_counts(), Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Constraints::constraintsRep, Constraints::shape(), and Constraints::sharedVarsData.

Referenced by Constraints::Constraints(), and Constraints::shape().

**void reshape ( size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons )**

reshape the linear/nonlinear constraint arrays

- Resizes the linear and nonlinear constraint arrays at the base class. Does NOT currently resize the derived bounds arrays.

References Constraints::constraintsRep, Constraints::continuousLowerBnds, Constraints::discreteIntLowerBnds, Constraints::discreteRealLowerBnds, Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearIneqConCoeffs, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::nonlinearEqConTargets, Constraints::nonlinearIneqConLowerBnds, Constraints::nonlinearIneqConUpperBnds, Constraints::numLinearEqCons, Constraints::numLinearIneqCons, Constraints::numNonlinearEqCons, Constraints::numNonlinearIneqCons, and Constraints::reshape().
void manage_linear_constraints ( const ProblemDescDB & problem_db ) [protected]

perform checks on user input, convert linear constraint coefficient input to matrices, and assign defaults

Convenience function called from derived class constructors. The number of variables active for applying linear constraints is currently defined to be the number of active continuous variables plus the number of active discrete variables (the most general case), even though very few optimizers can currently support mixed variable linear constraints.

References Dakota::abort_handler(), Constraints::continuousLowerBnds, Dakota::copy_data(), Constraints::discreteIntLowerBnds, Constraints::discreteRealLowerBnds, ProblemDescDB::get_rv(), Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearIneqConCoeffs, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::numLinearEqCons, and Constraints::numLinearIneqCons.

Referenced by Constraints::Constraints().

Constraints * get_constraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd ) [private]

Used only by the constructor to initialize constraintsRep to the appropriate derived type.

Initializes constraintsRep to the appropriate derived type, as given by the variables view.

References SharedVariablesData::view().

Referenced by Constraints::Constraints(), and Constraints::copy().

Constraints * get_constraints ( const SharedVariablesData & svd ) const [private]

Used by copy() to initialize constraintsRep to the appropriate derived type.

Initializes constraintsRep to the appropriate derived type, as given by the variables view. The default derived class constructors are invoked.

References SharedVariablesData::view().

The documentation for this class was generated from the following files:

- DakotaConstraints.hpp
- DakotaConstraints.cpp

13.26 DataEnvironment Class Reference

Handle class for environment specification data.

Public Member Functions

- DataEnvironment ()
  constructor
- DataEnvironment (const DataEnvironment &)
  copy constructor
- ~DataEnvironment ()
  destructor
- DataEnvironment & operator= (const DataEnvironment &)
  assignment operator
- void write (std::ostream &s) const
  write a DataEnvironment object to an std::ostream
13.27. DATAENVIRONMENTREP CLASS REFERENCE

- void read (MPIUnpackBuffer &s)
  read a DataEnvironment object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  write a DataEnvironment object to a packed MPI buffer
- DataEnvironmentRep * data_rep ()
  return dataEnvRep

Private Attributes

- DataEnvironmentRep * dataEnvRep
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

13.26.1 Detailed Description

Handle class for environment specification data.

The DataEnvironment class is used to provide a memory management handle for the data in DataEnvironmentRep. It is populated by IDRProblemDescDB::environment_kwhandler() and is queried by the ProblemDescDB::get<datatype>() functions. A single DataEnvironment object is maintained in ProblemDescDB::environment-Spec.

The documentation for this class was generated from the following files:

- DataEnvironment.hpp
- DataEnvironment.cpp

13.27 DataEnvironmentRep Class Reference

Body class for environment specification data.

Public Attributes

- bool checkFlag
  flag for whether to run in check only mode (default false)
- String outputFile
  file name for output redirection (overrides command-line)
- String errorFile
  file name for error redirection (overrides command-line)
- String readRestart
  file name for restart read (overrides command-line)
- int stopRestart
  record at which to stop reading restart
- String writeRestart
file name for restart write (overrides command-line)

- bool preRunFlag
  flags invocation with command line option -pre_run
- bool runFlag
  flags invocation with command line option -run
- bool postRunFlag
  flags invocation with command line option -post_run

- String preRunInput
  filename for pre_run input
- String preRunOutput
  filename for pre_run output
- String runInput
  filename for run input
- String runOutput
  filename for run output
- String postRunInput
  filename for post_run input
- String postRunOutput
  filename for post_run output

- unsigned short preRunOutputFormat
  tabular format for pre_run output
- unsigned short postRunInputFormat
  tabular format for post_run input

- bool graphicsFlag
  flags use of graphics by the environment (from the graphics specification in EnvIndControl)
- bool tabularDataFlag
  flags tabular data collection by the environment (from the tabular_graphics_data specification in EnvIndControl)

- String tabularDataFile
  the filename used for tabular data collection by the environment (from the tabular_graphics_file specification in EnvIndControl)

- unsigned short tabularFormat
  format for tabular data files (see enum)

- int outputPrecision
  output precision for tabular and screen output

- bool resultsOutputFlag
  flags use of results output to default file

- String resultsOutputFile
  named file for results output

- String topMethodPointer
  method identifier for the environment (from the top_method_pointer specification)
### Private Member Functions

- **DataEnvironmentRep ()**
  - *constructor*
- **∼DataEnvironmentRep ()**
  - *destructor*
- void *write* (std::ostream &s) const
  - *write a DataEnvironmentRep object to an std::ostream*
- void *read* (MPIUnpackBuffer &s)
  - *read a DataEnvironmentRep object from a packed MPI buffer*
- void *write* (MPIPackBuffer &s) const
  - *write a DataEnvironmentRep object to a packed MPI buffer*

### Private Attributes

- int *referenceCount*
  - *number of handle objects sharing this dataEnvironmentRep*

### Friends

- class **DataEnvironment**
  - *the handle class can access attributes of the body class directly*

---

### 13.27.1 Detailed Description

Body class for environment specification data.

The **DataEnvironmentRep** class is used to contain the data from the environment keyword specification. Default values are managed in the **DataEnvironmentRep** constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::environmentSpec is private.

The documentation for this class was generated from the following files:

- DataEnvironment.hpp
- DataEnvironment.cpp

---

### 13.28 DataFitSurrBasedLocalTraits Class Reference

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

Inheritance diagram for DataFitSurrBasedLocalTraits:

```
TraitsBase
 |
|---|
| DataFitSurrBasedLocalTraits
```
Public Member Functions

- `DataFitSurrBasedLocalTraits()`
  - *default constructor*
- `virtual ~DataFitSurrBasedLocalTraits()`
  - *destructor*
- `virtual bool is_derived()`
  - *A temporary query used in the refactor.*
- `bool supports_continuous_variables()`
  - *Return the flag indicating whether method supports continuous variables.*
- `bool supports_linear_equality()`
  - *Return the flag indicating whether method supports linear equalities.*
- `bool supports_linear_inequality()`
  - *Return the flag indicating whether method supports linear inequalities.*
- `bool supports_nonlinear_equality()`
  - *Return the flag indicating whether method supports nonlinear equalities.*
- `bool supports_nonlinear_inequality()`
  - *Return the flag indicating whether method supports nonlinear inequalities.*

13.28.1 Detailed Description

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

This minimizer uses a `SurrogateModel` to perform minimization based on local, global, or hierarchical surrogates. It achieves provable convergence through the use of a sequence of trust regions and the application of surrogate corrections at the trust region centers. A version of `TraitsBase` specialized for local surrogate-based minimizer.

The documentation for this class was generated from the following file:

- `DataFitSurrBasedLocalMinimizer.hpp`

13.29 DataFitSurrModel Class Reference

Derived model class within the surrogate model branch for managing data fit surrogates (global and local).

Inheritance diagram for DataFitSurrModel:

```
    Model
     |   
     v
SurrogateModel
     |   
     v
DataFitSurrModel
```
Public Member Functions

- **DataFitSurrModel (ProblemDescDB &problem_db)**
  
  *constructor*

- **DataFitSurrModel (Iterator &dace_iterator, Model &actual_model, const ActiveSet &set, const String &approx_type, const UShortArray &approx_order, short corr_type, short corr_order, short data_order, short output_level, const String &point_reuse, const String &import_build_points_file=String(), unsigned short import_build_format=TABULAR_ANNOTATED, bool import_build_active_only=false, const String &export_approx_points_file=String(), unsigned short export_approx_format=TABULAR_ANNOTATED)**
  
  *alternate constructor for instantiations on the fly*

- **~DataFitSurrModel ()**
  
  *destructor*

- **void total_points (int points)**
  
  *set pointsTotal and pointsManagement mode*

Protected Member Functions

- **DiscrepancyCorrection & discrepancy_correction ()**
  
  *return the DiscrepancyCorrection object used by SurrogateModels*

- **short correction_type ()**
  
  *return the correction type from the DiscrepancyCorrection object used by SurrogateModels*

- **bool initialize_mapping (ParLevLIter pl_iter)**
  
  *Perform any global updates prior to individual evaluate() calls.*

- **bool finalize_mapping ()**
  
  *restore state in preparation for next initialization*

- **void derived_evaluate (const ActiveSet &set)**
  
  *portion of evaluate() specific to DataFitSurrModel*

- **void derived_evaluate_nowait (const ActiveSet &set)**
  
  *portion of evaluate_nowait() specific to DataFitSurrModel*

- **const IntResponseMap & derived_synchronize ()**
  
  *portion of synchronize() specific to DataFitSurrModel*

- **const IntResponseMap & derived_synchronize_nowait ()**
  
  *portion of synchronize_nowait() specific to DataFitSurrModel*

- **Iterator & subordinate_iterator ()**
  
  *return daceIterator*

- **Model & surrogate_model ()**
  
  *return this model instance*

- **Model & truth_model ()**
  
  *return actualModel*

- **void derived_subordinate_models (ModelList &ml, bool recurse_flag)**
  
  *return actualModel (and optionally its sub-models)*

- **void update_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())**
  
  *pass request to actualModel if recursing and then update from it*

- **Interface & derived_interface ()**
  
  *return approxInterface*
• void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into actualModel
• void surrogate_response_mode (short mode)
  set responseMode and pass any bypass request on to actualModel for any lower-level surrogates.
• void surrogate_function_indices (const IntSet &surr_fn_indices)
  (re)set the surrogate index set in SurrogateModel::surrogateFnIndices and ApproximationInterface::approxFnIndices
• void build_approximation (size_t index=_NPOS)
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points.
• bool build_approximation (const Variables &vars, const IntResponsePair &response_pr, size_t index=_NPOS)
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the passed vars/response anchor point.
• void update_approximation (bool rebuild_flag, size_t index=_NPOS)
  replaces the approximation data with daceIterator results and rebuilds the approximation if requested
• void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag, size_t index=_NPOS)
  replaces the anchor point, and rebuilds the approximation if requested
• void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  replaces the current points array and rebuilds the approximation if requested
• void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  replaces the current points array and rebuilds the approximation if requested
• void append_approximation (bool rebuild_flag, size_t index=_NPOS)
  appends daceIterator results to a global approximation and rebuilds it if requested
• void append_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag, size_t index=_NPOS)
  appends a point to a global approximation and rebuilds it if requested
• void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  appends an array of points to a global approximation and rebuilds it if requested
• void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  appends a matrix of points to a global approximation and rebuilds it if requested
• void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
  remove approximation data added on previous append_approximation() call or a specified number of points
• void push_approximation ()
  retrieve a previous approximation data state
• bool push_available ()
  query for whether a trial increment can be retrieved
• void finalize_approximation ()
  finalize data fit by applying all previous trial increments
• void store_approximation (size_t index=_NPOS)
  store the current data fit approximation for later combination
• void restore_approximation (size_t index=_NPOS)
  restore a previous data fit approximation
• void remove_stored_approximation (size_t index=_NPOS)
  store the current data fit approximation for later combination
• void combine_approximation ()
  combine the current data fit approximation with one previously stored
• void clear_stored ()
  clear data stored in the approxInterface
• void run_dace_iterator (bool rebuild_flag, size_t index=_NPOS)
  execute the DACE iterator, append the approximation data, and rebuild the approximation if indicated
• SharedApproxData & shared_approximation ()
  retrieve the SharedApproxData from approxInterface
• std::vector< Approximation > & approximations ()
  retrieve the set of Approximations from approxInterface
• const RealVectorArray & approximation_coefficients (bool normalized=false)
  return the approximation coefficients from each Approximation (request forwarded to approxInterface)
• void approximation_coefficients (const RealVectorArray &approx_coeffs, bool normalized=false)
  set the approximation coefficients within each Approximation (request forwarded to approxInterface)
• const RealVector & approximation_variances (const Variables &vars)
  return the approximation variance from each Approximation (request forwarded to approxInterface)
• const Pecos::SurrogateData & approximation_data (size_t index)
  return the approximation data from a particular Approximation (request forwarded to approxInterface)
• void component_parallel_mode (short mode)
  update component parallel mode for supporting parallelism in actualModel
• IntIntPair estimate_partition_bounds (int max_eval_concurrency)
  estimate the minimum and maximum partition sizes that can be utilized by this Model
• void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set up actualModel for parallel operations
• void derived_init_serial ()
  set up actualModel for serial operations.
• void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration within actualModel
• void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the DataFitSurrModel (request forwarded to actualModel)
• void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)
  Service actualModel job requests received from the master. Completes when a termination message is received from stop_servers().
• void stop_servers ()
  Executed by the master to terminate actualModel server operations when DataFitSurrModel iteration is complete.
• void inactive_view (short view, bool recurse_flag=true)
  update the Model’s inactive view based on higher level (nested) context and optionally recurse into actualModel
• const String & interface_id () const
  return the approxInterface identifier
• bool evaluation_cache (bool recurse_flag=true) const
  if recurse_flag, return the actualModel evaluation cache usage
• bool restart_file (bool recurse_flag=true) const
  if recurse_flag, return the actualModel restart file usage
• void set_evaluation_reference ()
  set the evaluation counter reference points for the DataFitSurrModel (request forwarded to approxInterface and actualModel)
• void fine_grained_evaluation_counts ()
  request fine-grained evaluation reporting within approxInterface and actualModel
• void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  print the evaluation summary for the DataFitSurrModel (request forwarded to approxInterface and actualModel)
• void warm_start_flag (const bool flag)
  set the warm start flag, including actualModel

Protected Attributes

• const bool exportSurrogate
  whether to export the surrogate to file
• const bool autoRefine
  whether to automatically refine the surrogate during the build phase
• const int maxIterations
  Maximum number of times to refine the surrogate.
• const int maxFuncEvals
  Maximum number of evaluations while refining a surrogate.
• const Real convergenceTolerance
  Convergence criterion, compared to CV score for specified metric.
• const int softConvergenceLimit
  Max number of iterations for which there is no average improvement.
• const String refineCVMetric
  Type of error metric to test for surrogate refinement convergence.
• const int refineCVFolds
  Number of cross validation folds for surrogate refinement.

Private Member Functions

• void import_points (unsigned short tabular_format, bool active_only)
  optionally read surrogate data points from provided file
• void initialize_export ()
  initialize file stream for exporting surrogate evaluations
• void finalize_export ()
  finalize file stream for exporting surrogate evaluations
• void export_point (int eval_id, const Variables &vars, const Response &resp)
  initialize file stream for exporting surrogate evaluations
• void derived_synchronize_approx (bool block, IntResponseMap &approx_resp_map_rekey)
  Common code for processing of approximate response maps shared by derived_synchronize() and derived_synchronize_nowait()
• void update_global ()
  Updates fit arrays for global approximations.
• void update_local_multipoint ()
  Updates fit arrays for local or multipoint approximations.
• void build_global ()
  Builds a global approximation using daceIterator.
• void build_local_multipoint ()
  Builds a local or multipoint approximation using actualModel.
• void refine_surrogate (size_t index=_NPOS)
  Refine the built surrogate until convergence criteria are met.
• void interface_build_approx (size_t index=_NPOS)
  Call build_approximation on the interface, passing appropriate constraints.
• void init_model (Model &model)
  update actualModel with data from constraints/labels/sets
• void update_model (Model &model)
  update actualModel with data from current variables/bounds
• void update_from_model (const Model &model)
  update current variables/labels/bounds/targets with data from actualModel
• bool consistent (const Variables &vars) const
  test if inactive state is consistent
• bool inside (const Variables &vars) const
  test if active vars are within [l_bnds, u_bnds]
• bool active_vars_compare (const Variables &vars, const Pecos::SurrogateDataVars &sdv) const
  test for exact equality in values between active vars and sdv

Private Attributes

• DiscrepancyCorrection deltaCorr
  manages construction and application of correction functions that are applied to a surrogate model (DataFitSurr or HierarchSurr) in order to reproduce high fidelity data.
• int pointsTotal
  total points the user specified to construct the surrogate
• short pointsManagement
  configuration for points management in build_global()
• String pointReuse
  type of point reuse for approximation builds: all, region (default if points file), or none (default if no points file)
• String importPointsFile
  file name from import_build_points_file specification
• String `exportPointsFile`
  
  file name from `export_approx_points_file specification`

• unsigned short `exportFormat`
  
  file export format for variables and approximate responses

• `std::ofstream exportFileStream`
  
  output file stream for `export_approx_points_file specification`

• Interface `approxInterface`
  
  manages the building and subsequent evaluation of the approximations (required for both global and local)

• `Model actualModel`
  
  the truth model which provides evaluations for building the surrogate (optional for global, required for local)

• `Iterator daceIterator`
  
  selects parameter sets on which to evaluate `actualModel` in order to generate the necessary data for building global approximations (optional for global since restart data may also be used)

### Additional Inherited Members

#### 13.29.1 Detailed Description

Derived model class within the surrogate model branch for managing data fit surrogates (global and local)

The `DataFitSurrModel` class manages global or local approximations (surrogates that involve data fits) that are used in place of an expensive model. The class contains an `approxInterface` (required for both global and local) which manages the approximate function evaluations, an `actualModel` (optional for global, required for local) which provides truth evaluations for building the surrogate, and a `daceIterator` (optional for global, not used for local) which selects parameter sets on which to evaluate `actualModel` in order to generate the necessary data for building global approximations.

#### 13.29.2 Constructor & Destructor Documentation

```cpp
~DataFitSurrModel() [inline]
```

destructor

Virtual destructor handles `referenceCount` at base `Model` level.

References `DataFitSurrModel::exportPointsFile`, and `DataFitSurrModel::finalize_export()`.

#### 13.29.3 Member Function Documentation

```cpp
bool finalize_mapping() [protected], [virtual]
```

restore state in preparation for next initialization

Inactive variables must be propagated when a `HierarchSurrModel` is employed by a sub-iterator (e.g., OUU with MLMC or MLPCE). In current use cases, this can occur once per sub-iterator execution within `Model::initialize_mapping()`.

Reimplemented from `Model`.

References `DataFitSurrModel::actualModel`, `SurrogateModel::finalize_mapping()`, and `Model::finalize_mapping()`.
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]

portion of evaluate() specific to DataFitSurrModel

Compute the response synchronously using actualModel, approxInterface, or both (mixed case). For the approxInterface portion, build the approximation if needed, evaluate the approximate response, and apply correction (if active) to the results.

Reimplemented from Model.

References Response::active_set(), DataFitSurrModel::actualModel, DiscrepancyCorrection::apply(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(), DataFitSurrModel::build_approximation(), DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), Response::copy(), Model::current_response(), Model::currentResponse, Model::currentVariables, DataFitSurrModel::delta_Corr, Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate(), DataFitSurrModel::export_point(), DataFitSurrModel::exportPointsFile, SurrogateModel::force_rebuild(), Model::hierarchicalTagging, Interface::map(), Model::outputLevel, ActiveSet::request_vector(), SurrogateModel::response_mapping(), SurrogateModel::surrModelEvalCntr, Response::update(), and DataFitSurrModel::update_model().

void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]

portion of evaluate_nowait() specific to DataFitSurrModel

Compute the response asynchronously using actualModel, approxInterface, or both (mixed case). For the approxInterface portion, build the approximation if needed and evaluate the approximate response in a quasi-asynchronous approach (ApproximationInterface::map() performs the map synchronously and bookkeeps the results for return in derived_synchronize() below).

Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(), DataFitSurrModel::build_approximation(), Variables::copy(), Model::currentResponse, Model::currentVariables, Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate_nowait(), Interface::evaluation_id(), Model::evaluation_id(), DataFitSurrModel::exportPointsFile, SurrogateModel::force_rebuild(), Model::hierarchicalTagging, Interface::map(), SurrogateModel::rawVarsMap, ActiveSet::request_vector(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrModelEvalCntr, SurrogateModel::truthIdMap, and DataFitSurrModel::update_model().

const IntResponseMap & derived_synchronize () [protected], [virtual]

portion of synchronize() specific to DataFitSurrModel

Blocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case). For the approxInterface portion, apply correction (if active) to each response in the array. derived_synchronize() is designed for the general case where derived_evaluate_nowait() may be inconsistent in its use of actual evaluations, approximate evaluations, or both.

Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::aggregate_response(), SurrogateModel::check_key(), DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), DataFitSurrModel::delta_Corr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, Model::rekey_synch(), SurrogateModel::response_mapping(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, and SurrogateModel::truthIdMap.

const IntResponseMap & derived_synchronize_nowait () [protected], [virtual]

portion of synchronize_nowait() specific to DataFitSurrModel
Nonblocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case). For the approxInterface portion, apply correction (if active) to each response in the map. derived_synchronize_nowait() is designed for the general case where derived_evaluate_nowait() may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from Model.

References Dakota::abort_handler(), DataFitSurrModel::actualModel, SurrogateModel::aggregate_response(), SurrogateModel::cachedApproxRespMap, DataFitSurrModel::component_parallel_model(), DiscrepancyCorrection::compute(), DataFitSurrModel::deltaCorr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, Model::rekey_synch(), SurrogateModel::response_mapping(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, and SurrogateModel::truthIdMap.

```cpp
void build_approximation ( size_t index = NPOS ) [protected], [virtual]
```

Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points.

This function constructs a new approximation, discarding any previous data. It constructs any required data for SurrogateData::{vars,resp}Data and does not define an anchor point for SurrogateData::anchor{Vars,Resp}, so is an unconstrained build.

Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, DataFitSurrModel::autoRefine, DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), Interface::clear_current(), DataFitSurrModel::interface_build_approx(), DataFitSurrModel::refine_surrogate(), Dakota::strbegins(), Model::surrogateType, DataFitSurrModel::update_global(), DataFitSurrModel::update_local_multipoint(), and DataFitSurrModel::update_model().

Referenced by DataFitSurrModel::derived_evaluate(), and DataFitSurrModel::derived_evaluate_nowait().

```cpp
bool build_approximation ( const Variables & vars, const IntResponsePair & response_pr, size_t index = NPOS ) [protected], [virtual]
```

Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the passed vars/response anchor point.

This function constructs a new approximation, discarding any previous data. It uses the passed data to populate SurrogateData::anchor{Vars,Resp} and constructs any required data points for SurrogateData::{vars,resp}Data. Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, DataFitSurrModel::autoRefine, DataFitSurrModel::build_global(), Interface::clear_current(), DataFitSurrModel::interface_build_approx(), DataFitSurrModel::refine_surrogate(), Dakota::strbegins(), Model::surrogateType, Interface::update_approximation(), DataFitSurrModel::update_global(), DataFitSurrModel::update_local_multipoint(), and DataFitSurrModel::update_model().

```cpp
void update_approximation ( bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

replaces the approximation data with daceIterator results and rebuilds the approximation if requested.

This function populates/replaces SurrogateData::anchor{Vars,Resp} and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::{vars,resp}Data) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Iterator::all_responses(), Iterator::all_samples(), Iterator::all_variables(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Iterator::compact_mode(), DataFitSurrModel::daceIterator, Model-
::numFns, Model::outputLevel, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

```cpp
void update_approximation ( const Variables & vars, const IntResponsePair & response, bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

replaces the anchor point, and rebuilds the approximation if requested

This function populates replaces SurrogateData::anchor{Vars,Resp} and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::{vars,resp}Data) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Model::outputLevel, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

```cpp
void update_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map, bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

replaces the current points array and rebuilds the approximation if requested

This function populates replaces SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Model::outputLevel, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

```cpp
void update_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map, bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

replaces the current points array and rebuilds the approximation if requested

This function populates replaces SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Model::outputLevel, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

```cpp
void append_approximation ( bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

appends daceIterator results to a global approximation and rebuilds it if requested

This function appends one point to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Iterator::all_responses(), Iterator::all_samples(), Iterator::all_variables(), Interface::append_approximation(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Iterator::compact_mode(), DataFitSurrModel::daceIterator, Model::numFns, Model::outputLevel, Interface::rebuild_approximation(), and Model::surrogateType.
Referenced by DataFitSurrModel::run_dace_iterator().

```cpp
void append_approximation ( const Variables & vars, const IntResponsePair & response_pr, bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

Appends a point to a global approximation and rebuilds it if requested.

This function appends one point to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Interface::append_approximation(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Model::outputLevel, Interface::rebuild_approximation(), and Model::surrogateType.

```cpp
void append_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map, bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

Appends an array of points to a global approximation and rebuilds it if requested.

This function appends multiple points to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Interface::append_approximation(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Model::outputLevel, Interface::rebuild_approximation(), and Model::surrogateType.

```cpp
void append_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map, bool rebuild_flag, size_t index = NPOS ) [protected], [virtual]
```

Appends a matrix of points to a global approximation and rebuilds it if requested.

This function appends multiple points to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Interface::append_approximation(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Model::numFns, Model::outputLevel, Interface::rebuild_approximation(), and Model::surrogateType.

```cpp
void derived_init communicators ( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag = true ) [protected], [virtual]
```

Sets up actualModel for parallel operations.

Asynchronous flags need to be initialized for the sub-models. In addition, max_eval_concurrency is the outer level iterator concurrency, not the DACE concurrency that actualModel will see, and recomputing the message-lengths on the sub-model is probably not a bad idea either. Therefore, recompute everything on actualModel using init_communicators.

Reimplemented from Model.

References DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, DataFitSurrModel::daceIterator, Model::derivative_concurrency(), ProblemDescDB::get_db_method_node(), ProblemDescDB::get_db_model_node(),
Iterator::initCommunicators(), Model::initCommunicators(), Iterator::is_null(), Model::is_null(), Iterator::iteratedModel(), Iterator::maximumEvaluationConcurrency(), Iterator::methodId(), Interface::minimumPoints(), Model::modelId(), Model::probDescDB, ProblemDescDB::setDbListNodes(), ProblemDescDB::setDbMethodNode(), and ProblemDescDB::setDbModelNodes().

```cpp
void import_points ( unsigned short tabular_format, bool active_only ) [private]
```

optionally read surrogate data points from provided file
Constructor helper to read the points file once, if provided, and then reuse its data as appropriate within build_global(). Surrogate data imports default to active/inactive variables, but user can override to active only process arrays of data from TabularIO::read_data_table() above
References DataFitSurrModel::actualModel, Response::copy(), Variables::copy(), Model::currentResponse(), Model::currentVariables(), Variables::cv(), Dakota::data_pairs, Variables::div(), Variables::drv(), Variables::dsv(), ParamResponsePair::evalId(), Model::evaluationCache(), DataFitSurrModel::importPointsFile, ParamResponsePair::interfaceId(), Model::interfaceId(), Model::is_null(), Model::numFns, Model::outputLevel, Model::parallelLib, Model::restart_file(), Variables::tv(), and ParallelLibrary::write_restart().

Referenced by DataFitSurrModel::DataFitSurrModel().

```cpp
void initialize_export ( ) [private]
```
initialize file stream for exporting surrogate evaluations
Constructor helper to export approximation-based evaluations to a file.
References Model::currentResponse, Model::currentVariables, DataFitSurrModel::exportFileStream, DataFitSurrModel::exportFormat, and DataFitSurrModel::exportPointsFile.

Referenced by DataFitSurrModel::DataFitSurrModel().

```cpp
void finalize_export ( ) [private]
```
finalize file stream for exporting surrogate evaluations
Constructor helper to export approximation-based evaluations to a file.
References DataFitSurrModel::exportFileStream, and DataFitSurrModel::exportPointsFile.

Referenced by DataFitSurrModel::~DataFitSurrModel().

```cpp
void export_point ( int eval_id, const Variables & vars, const Response & resp ) [private]
```
initialize file stream for exporting surrogate evaluations
Constructor helper to export approximation-based evaluations to a file. Exports all variables, so it's clear at what values of inactive it was built at
References DataFitSurrModel::exportFileStream, DataFitSurrModel::exportFormat, DataFitSurrModel::interfaceId(), Model::iteratorSpaceToUserSpace(), and Model::recastings().

Referenced by DataFitSurrModel::derived_evaluate(), and DataFitSurrModel::derived_synchronize_approx().

```cpp
void build_global ( ) [private]
```
Builds a global approximation using daceIterator.
Determines points to use in building the approximation and then evaluate them on actualModel using daceIterator. Any changes to the bounds should be performed by setting them at a higher level (e.g., SurrBasedOptStrategy).
References Dakota::abort_handler(), DataFitSurrModel::active_vars_compare(), DataFitSurrModel::activeModel, Interface::append_approximation(), Interface::approximation_data(), DataFitSurrModel::approxInterface, DataFitSurrModel::component_parallel_mode(), DataFitSurrModel::consistent(), Model::currentVariables, Variables::cv(), Model::cv(), DataFitSurrModel::daceIterator, Dakota::data_pairs, Variables::div(), Model::div(), Variables::drv(), Model::drv(), DataFitSurrModel::inside(), Model::interface_id(), Iterator::is_null(), Model::is_null(), Interface::minimum_points(), Iterator::num_samples(), Model::outputLevel, DataFitSurrModel::pointReuse, DataFitSurrModel::pointsManagement, DataFitSurrModel::pointsTotal, Model::recastings(), Interface::recommended_points(), DataFitSurrModel::run_dace_iterator(), Iterator::sampling_reset(), SurrogateModel::surrogateFnIndices, and Model::user_space_to_iterator_space().

Referenced by DataFitSurrModel::build_approximation().

**void build_local_multipoint( ) [private]**

Builds a local or multipoint approximation using actualModel.

Evaluate the value, gradient, and possibly Hessian needed for a local or multipoint approximation using actualModel.

References Response::active_set(), DataFitSurrModel::activeModel, DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(), DataFitSurrModel::component_parallel_mode(), Model::continuous_variable_ids(), Model::current_response(), Model::current_variables(), ActiveSet::derivative_vector(), Model::evaluate(), Model::evaluation_id(), Model::hessian_type(), Model::numFns, ActiveSet::request_vector(), Dakota::strbegins(), Model::surrogateType, and Interface::update_approximation().

Referenced by DataFitSurrModel::build_approximation().

**void init_model( Model & model ) [private]**

update actualModel with data from constraints/labels/sets

Update variables and constraints data within model using values and labels from currentVariables and bound/linear/nonlinear constraints from userDefinedConstraints.

References Dakota::abort_handler(), Model::aleatDistParams, Model::aleatory_distribution_parameters(), Variables::all_continuous_variable_labels(), Model::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Model::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Model::all_discrete_real_variable_labels(), SurrogateModel::approxBuilds, Variables::continuous_variable_labels(), Model::continuous_variable_labels(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Variables::cv(), Model::cv(), Model::design_set_int_values(), Model::design_set_real_values(), Variables::discrete_int_variable_labels(), Model::discrete_int_variable_labels(), Variables::discrete_real_variable_labels(), Model::discrete_real_variable_labels(), Model::discrete_state_set_int_values(), Model::discrete_state_set_real_values(), Model::discreteDesignSetIntValues, Model::discreteDesignSetRealValues, Model::discreteStateSetRealValues, Variables::div(), Model::div(), Variables::drv(), Model::drv(), Model::epistDistParams, Model::epistemic_distribution_parameters(), Response::function_labels(), SharedVariablesData::id(), Variables::inactive_continuous_variable_labels(), Model::inactive_continuous_variable_labels(), Variables::inactive_discrete_int_variable_labels(), Model::inactive_discrete_int_variable_labels(), Variables::inactive_discrete_real_variable_labels(), Model::inactive_discrete_real_variable_labels(), Model::is_null(), Constraints::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_coeffs(), Constraints::linear_eq_constraint_targets(), Model::linear_eq_constraint_targets(), Constraints::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_lower_bounds(), Constraints::linear_ineq_constraint_upper_bounds(), Model::linear_ineq_constraint_upper_bounds(), Constraints::nonlinear_eq_constraint_coeffs(), Model::nonlinear_eq_constraint_coeffs(), Constraints::nonlinear_eq_constraint_lower_bounds(), Model::nonlinear_eq_constraint_lower_bounds(), Constraints::nonlinear_eq_constraint_upper_bounds(), Model::nonlinear_eq_constraint_upper_bounds(), Constraints::num_linear_eq_constraints(), Constraints::num_linear_ineq_constraints(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Constraints::num_linear_ineq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Constraints::num_linear_eq_constraints(), Constraints::num_linear_ineq_constraints(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints().
void update_model ( Model & model ) [private]
update actualModel with data from current variables/bounds

Update variables and constraints data within model using values and labels from currentVariables and bound/linear/nonlinear constraints from userDefinedConstraints.

References Dakota::abort_handler(), Constraints::all_continuous_lower_bounds(), Model::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Model::all_continuous_upper_bounds(), Variables::all_continuous_variables(), Model::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Model::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), Model::all_discrete_int_upper_bounds(), Variables::all_discrete_int_variables(), Model::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Model::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_upper_bounds(), Model::all_discrete_real_upper_bounds(), Variables::all_discrete_real_variables(), Model::all_discrete_real_variables(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variables(), Model::continuous_variables(), Model::currentVariables, Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Variables::discrete_int_variables(), Model::discrete_int_variables(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_upper_bounds(), Model::discrete_real_upper_bounds(), Variables::discrete_real_variables(), Model::discrete_real_variables(), Model::is_null(), Model::userDefinedConstraints, and Variables::view().

Referenced by DataFitSurrModel::initialize_mapping(), and DataFitSurrModel::derived_evaluate(), and DataFitSurrModel::derived_evaluate_nowait().

void update_from_model ( const Model & model ) [private]
update current variables/labels/bounds/targets with data from actualModel

Update values and labels in currentVariables and bound/linear/nonlinear constraints in userDefinedConstraints from variables and constraints data within model.

References Dakota::abort_handler(), Model::aleatDistParams, Model::aleatory_distribution_parameters(), Constraints::all_continuous_lower_bounds(), Model::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Model::all_continuous_upper_bounds(), Variables::all_continuous_variables(), Model::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Model::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), Model::all_discrete_int_upper_bounds(), Variables::all_discrete_int_variables(), Model::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Model::all_discrete_real_lower_bounds(), Variables::all_discrete_real_variables(), Model::all_discrete_real_variables(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variables(), Model::currentVariables, Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Variables::discrete_int_variables(), Model::discrete_int_variables(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_upper_bounds(), Model::discrete_real_upper_bounds(), Variables::discrete_real_variables(), Model::discrete_real_variables(), Variables::all_discrete_string_variable_labels(), Model::all_discrete_string_variables(), SurrogateModel::approxBuilds, Model::current_variables(), Model::currentResponse, Model::currentVariables, Variables::cv(), Model::cv(), Model::discrete_design_set_int_values(), Model::discrete_design_set_real_values(), Model::discrete_design_set_string_values(), Model::discrete_state_set_int_values(), Model::discrete_state_set_string_values(), Model::discreteDesignSetIntValues, Model::discreteDesignSetRealValues, Model::discreteDesignSetStringValue, Model::discreteStateSetIntValues, Model::discreteStateSetRealValues, Model::discreteStateSetStringValue, Variables::div(), Model::div(), Variables::drv(), Model::epistDistParams, Model::epistemic_distribution_parameters(), Response::function_labels(), SharedVariablesData::id(), Constraints::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_coeffs(), Constraints-
CHAPTER 13. CLASS DOCUMENTATION

::linear_eq_constraint_targets(), Model::linear_eq_constraint_targets(), Constraints::linear_ineq_constraint_coefs(), Model::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_coeffs(), Constraints::nonlinear_ineq_constraint_coeffs(), Model::nonlinear_ineq_constraint_targets(), Constraints::nonlinear_ineq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::linear_eq_constraints(), Model::linear_ineq_constraints(), Model::nonlinear_eq_constraints(), Model::nonlinear_ineq_constraints(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Variables::shared_data(), and Model::userDefinedConstraints.

Referenced by DataFitSurrModel::DataFitSurrModel(), and DataFitSurrModel::update_from_subordinate_model().

13.29.4 Member Data Documentation

Model actualModel [private]

the truth model which provides evaluations for building the surrogate (optional for global, required for local)

actualModel is unrestricted in type; arbitrary nestings are possible.

Referenced by DataFitSurrModel::build_approximation(), DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), DataFitSurrModel::consistent(), DataFitSurrModel::DataFitSurrModel(), DataFitSurrModel::evaluate(), DataFitSurrModel::evaluate_nowait(), DataFitSurrModel::free_communicators(), DataFitSurrModel::init_communicators(), DataFitSurrModel::init_serial(), DataFitSurrModel::set_communicators(), DataFitSurrModel::subordinate_models(), DataFitSurrModel::sync_nowait(), DataFitSurrModel::sync_partition_bounds(), DataFitSurrModel::compute_partition_bounds(), DataFitSurrModel::interface_build_approx(), DataFitSurrModel::interface_build_approx(), DataFitSurrModel::primary_response_fn_sense(), DataFitSurrModel::primary_response_fn_weights(), DataFitSurrModel::primary_response_fn_sense(), DataFitSurrModel::primary_response_fn_weights(), DataFitSurrModel::primary_response_fn_sense(), DataFitSurrModel::primary_response_fn_weights(), DataFitSurrModel::serve_run(), DataFitSurrModel::stop_servers(), DataFitSurrModel::surrogate_response_mode(), DataFitSurrModel::truth_model(), DataFitSurrModel::update_from_subordinate_model(), DataFitSurrModel::update_global(), DataFitSurrModel::update_local_multipoint(), and DataFitSurrModel::warm_start_flag().

The documentation for this class was generated from the following files:

- DataFitSurrModel.hpp
- DataFitSurrModel.cpp

13.30 DataInterface Class Reference

Handle class for interface specification data.

Public Member Functions

- DataInterface ()
  constructor
- DataInterface (const DataInterface &)
  copy constructor
- ~DataInterface ()
  destructor
- DataInterface & operator= (const DataInterface &)
  assignment operator
13.31. **DATAMETHOD CLASS REFERENCE**

- **void write** (std::ostream &s) const
  
  *write a DataInterface object to an std::ostream*

- **void read (MPIUnpackBuffer &s)**
  
  *read a DataInterface object from a packed MPI buffer*

- **void write (MPIPackBuffer &s) const**
  
  *write a DataInterface object to a packed MPI buffer*

- **DataInterfaceRep * data_rep ()**
  
  *return dataIfaceRep*

**Static Public Member Functions**

- **static bool id_compare (const DataInterface &di, const std::string &id)**
  
  *compares the idInterface attribute of DataInterface objects*

**Private Attributes**

- **DataInterfaceRep * dataIfaceRep**
  
  *pointer to the body (handle-body idiom)*

**Friends**

- class **ProblemDescDB**

- class **NIDRProblemDescDB**

13.30.1 **Detailed Description**

Handle class for interface specification data.

The **DataInterface** class is used to provide a memory management handle for the data in DataInterface-Rep. It is populated by IDRProblemDescDB::interface_kwhandler() and is queried by the ProblemDescDB::get_-<datatype>() functions. A list of DataInterface objects is maintained in ProblemDescDB::dataInterfaceList, one for each interface specification in an input file.

The documentation for this class was generated from the following files:

- DataInterface.hpp
- DataInterface.cpp

13.31 **DataMethod Class Reference**

Handle class for method specification data.

**Public Member Functions**

- **DataMethod ()**
  
  *constructor*

- **DataMethod (const DataMethod &)**
  
  *copy constructor*

- **~DataMethod ()**


destructor

• DataMethod & operator= (const DataMethod &)

assignment operator

• void write (std::ostream &s) const

write a DataMethod object to an std::ostream

• void read (MPIUnpackBuffer &s)

read a DataMethod object from a packed MPI buffer

• void write (MPIPackBuffer &s) const

write a DataMethod object to a packed MPI buffer

• DataMethodRep * data_rep ()

return dataMethodRep

Static Public Member Functions

• static bool id_compare (const DataMethod &dm, const std::string &id)

compares the idMethod attribute of DataMethod objects

Private Attributes

• DataMethodRep * dataMethodRep

pointer to the body (handle-body idiom)

Friends

• class ProblemDescDB

• class NIDRProblemDescDB

13.31.1 Detailed Description

Handle class for method specification data.

The DataMethod class is used to provide a memory management handle for the data in DataMethodRep. It is populated by IDRProblemDescDB::methodHandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataMethod objects is maintained in ProblemDescDB::dataMethodList, one for each method specification in an input file.

The documentation for this class was generated from the following files:

• DataMethod.hpp

• DataMethod.cpp

13.32 DataMethodRep Class Reference

Body class for method specification data.
Public Attributes

- **String idMethod**
  - string identifier for the method specification data set (from the id-method specification in MethodIndControl)
- **String modelPointer**
  - string pointer to the model specification to be used by this method (from the model_pointer specification in MethodIndControl)
- **String lowFidModelPointer**
  - string to point to the low fidelity model for Bayesian experimental design
- **short methodOutput**
  - method verbosity control: \{SILENT, QUIET, NORMAL, VERBOSE, DEBUG\} \_OUTPUT (from the output specification in MethodIndControl)
- **int maxIterations**
  - maximum number of iterations allowed for the method (from the max_iterations specification in MethodIndControl)
- **int maxRefineIterations**
  - maximum number of refinement iterations allowed for a uniform/adaptive refinement approach (from the max_refinement_iterations specification in MethodIndControl)
- **int maxSolverIterations**
  - maximum number of internal solver iterations allowed for the method (from the max_solver_iterations specification in MethodIndControl)
- **int maxFunctionEvaluations**
  - maximum number of function evaluations allowed for the method (from the max_function_evaluations specification in MethodIndControl)
- **bool speculativeFlag**
  - flag for use of speculative gradient approaches for maintaining parallel load balance during the line search portion of optimization algorithms (from the speculative specification in MethodIndControl)
- **bool methodUseDerivsFlag**
  - flag for usage of derivative data to enhance the computation of surrogate models (PCE/SC expansions, GP models for EGO/EGRA/EGIE) based on the use_derivatives specification
- **Real convergenceTolerance**
  - iteration convergence tolerance for the method (from the convergence_tolerance specification in MethodIndControl)
- **Real constraintTolerance**
  - tolerance for controlling the amount of infeasibility that is allowed before an active constraint is considered to be violated (from the constraint_tolerance specification in MethodIndControl)
- **bool methodScaling**
  - flag indicating scaling status (from the scaling specification in MethodIndControl)
- **size\_t numFinalSolutions**
  - number of final solutions returned from the iterator
- **unsigned short methodName**
  - the method selection: one of the optimizer, least squares, nond, dace, or parameter study methods
- **unsigned short subMethod**
  - enum value for a sub-method type
- **String subMethodName**
  - string identifier for a sub-method name within a multi-option method specification (e.g., from meta-iterators)
• String subModelPointer
  
  string pointer for a sub-model specification used by a meta-iterator

• String subMethodPointer
  
  string pointer for a sub-method specification used by a meta-iterator

• int iteratorServers
  
  number of servers for concurrent iterator parallelism (from the iterator_servers specification)

• int procsPerIterator
  
  number of processors for each concurrent iterator partition (from the processors_per_iterator specification)

• short iteratorScheduling
  
  type of scheduling ({DEFAULT,MASTER,PEER} SCHEDULING) used in concurrent iterator parallelism (from the iterator_scheduling specification)

• StringArray hybridMethodNames
  
  array of methods for the sequential and collaborative hybrid meta-iterators (from the method_name_list specification)

• StringArray hybridModelPointers
  
  array of models for the sequential and collaborative hybrid meta-iterators (from the model_pointer_list specification)

• StringArray hybridMethodPointers
  
  array of methods for the sequential and collaborative hybrid meta-iterators (from the method_pointer_list specification)

• String hybridGlobalMethodName
  
  global method name for embedded hybrids (from the global_method_name specification)

• String hybridGlobalModelPointer
  
  global model pointer for embedded hybrids (from the global_model_pointer specification)

• String hybridGlobalMethodPointer
  
  global method pointer for embedded hybrids (from the global_method_pointer specification)

• String hybridLocalMethodName
  
  local method name for embedded hybrids (from the local_method_name specification)

• String hybridLocalModelPointer
  
  local model pointer for embedded hybrids (from the local_model_pointer specification)

• String hybridLocalMethodPointer
  
  local method pointer for embedded hybrids (from the local_method_pointer specification)

• Real hybridLSProb
  
  local search probability for embedded hybrids (from the local_search_probability specification)

• int concurrentRandomJobs
  
  number of random jobs to perform in the pareto_set and multi_start meta-iterators (from the random_starts and random_weight_sets specifications)

• RealVector concurrentParameterSets
  
  user-specified (i.e., nonrandom) parameter sets to evaluate in the pareto_set and multi_start meta-iterators (from the starting_points and weight_sets specifications)

• unsigned short softConvLimit
  
  number of consecutive iterations with change less than convergenceTolerance required to trigger convergence

• bool surrBasedLocalLayerBypass
flag to indicate user-specification of a bypass of any/all layerings in evaluating truth response values in SBL.

- **RealVector trustRegionInitSize**
  initial trust region sizes in the surrogate-based local method (from the initial_size specification in MethodSBL), one size per surrogate model (notes: no trust region for the truth model; sizes are relative values, e.g., 0.1 = 10% of range of global bounds for each variable

- **Real trustRegionMinSize**
  minimum trust region size in the surrogate-based local method (from the minimum_size specification in MethodSBL), if the trust region size falls below this threshold the SBL iterations are terminated (note: if kriging is used with SBL, the min trust region size is set to 1.0e-3 in attempt to avoid ill-conditioned matrixes that arise in kriging over small trust regions)

- **Real trustRegionContractTrigger**
  trust region minimum improvement level (ratio of actual to predicted decrease in objective fcn) in the surrogate-based local method (from the contract_threshold specification in MethodSBL), the trust region shrinks or is rejected if the ratio is below this value ("eta_1" in the Conn-Gould-Toint trust region book)

- **Real trustRegionExpandTrigger**
  trust region sufficient improvement level (ratio of actual to predicted decrease in objective fn) in the surrogate-based local method (from the expand_threshold specification in MethodSBL), the trust region expands if the ratio is above this value ("eta_2" in the Conn-Gould-Toint trust region book)

- **Real trustRegionContract**
  trust region contraction factor in the surrogate-based local method (from the contract_factor specification in MethodSBL)

- **Real trustRegionExpand**
  trust region expansion factor in the surrogate-based local method (from the expansion_factor specification in MethodSBL)

- **short surrBasedLocalSubProbObj**
  SBL approximate subproblem objective: ORIGINAL_PRIMARY, SINGLE_OBJECTIVE, LAGRANGIAN_OBJECTIVE, or AUGMENTED_LAGRANGIAN_OBJECTIVE.

- **short surrBasedLocalSubProbCon**
  SBL approximate subproblem constraints: NO_CONSTRAINTS, LINEARIZED_CONSTRAINTS, or ORIGINAL_CONSTRAINTS.

- **short surrBasedLocalMeritFn**
  SBL merit function type: BASIC_PENALTY, ADAPTIVE_PENALTY, BASIC_LAGRANGIAN, or AUGMENTED_LAGRANGIAN.

- **short surrBasedLocalAcceptLogic**
  SBL iterate acceptance logic: TR_RATIO or FILTER.

- **short surrBasedLocalConstrRelax**
  SBL constraint relaxation method: NO_RELAX or HOMOTOPY.

- **bool surrBasedGlobalReplacePts**
  user-specified method for adding points to the set upon which the next surrogate is based in the surrogate-based global method.

- **String dlDetails**
  string of options for a dynamically linked solver

- **void * dlLib**
  handle to dynamically loaded library

- **int verifyLevel**
  the verify_level specification in MethodNPSOLDC
Real functionPrecision

the function_precision specification in MethodNPSOLDC and the EPSILON specification in NOMAD

Real lineSearchTolerance

the linesearch_tolerance specification in MethodNPSOLDC

Real absConvTol

absolute function convergence tolerance

Real xConvTol

x-convergence tolerance

Real singConvTol

singular convergence tolerance

Real singRadius

radius for singular convergence test

Real falseConvTol

false-convergence tolerance

Real initTRRadius

initial trust radius

int covarianceType

kind of covariance required

bool regressDiag

whether to print the regression diagnostic vector

String searchMethod

the search_method specification for Newton and nonlinear interior-point methods in MethodOPTPPDC

Real gradientTolerance

the gradient_tolerance specification in MethodOPTPPDC

Real maxStep

the max_step specification in MethodOPTPPDC

short meritFn

the merit_function specification for nonlinear interior-point methods in MethodOPTPPDC

Real stepLenToBoundary

the steplength_to_boundary specification for nonlinear interior-point methods in MethodOPTPPDC

Real centeringParam

the centering_parameter specification for nonlinear interior-point methods in MethodOPTPPDC

int searchSchemeSize

the search_scheme_size specification for PDS methods in MethodOPTPPDC

Real initStepLength

the initStepLength choice for nonlinearly constrained APPS in MethodAPPSDC

Real contractStepLength

the contractStepLength choice for nonlinearly constrained APPS in MethodAPPSDC

Real threshStepLength

the threshStepLength choice for nonlinearly constrained APPS in MethodAPPSDC

String meritFunction

the meritFunction choice for nonlinearly constrained APPS in MethodAPPSDC

Real constrPenalty
the constrPenalty choice for nonlinearly constrained APPS in MethodAPPSDC

- Real smoothFactor
  the initial smoothFactor value for nonlinearly constrained APPS in MethodAPPSDC

- Real constraintPenalty
  the initial constraint_penalty for COLINY methods in MethodAPPS, MethodSCOLIBDIR, MethodSCOLIBPS, MethodSCOLIBSW and MethodSCOLIBEA

- bool constantPenalty
  the constant_penalty flag for COLINY methods in MethodSCOLIBPS and MethodSCOLIBSW

- Real globalBalanceParam
  the global_balance parameter for the DIRECT method in MethodSCOLIBDIR

- Real localBalanceParam
  the local_balance parameter for the DIRECT method in MethodSCOLIBDIR

- Real maxBoxSize
  the max_boxsize_limit for the DIRECT method in MethodSCOLIBDIR

- Real minBoxSize
  the min_boxsize_limit for the DIRECT method in MethodSCOLIBDIR and MethodNCSUDC

- String boxDivision
  the division setting (major_dimension or all_dimensions) for the DIRECT method in MethodSCOLIBDIR

- bool mutationAdaptive
  the non_adaptive specification for the coliny_ea method in MethodSCOLIBEA

- bool showMiscOptions
  the show_misc_options specification in MethodSCOLIBDC

- StringArray miscOptions
  the misc_options specification in MethodSCOLIBDC

- Real solnTarget
  the solution_target specification in MethodSCOLIBDC

- Real crossoverRate
  the crossover_rate specification for EA methods in MethodSCOLIBEA

- Real mutationRate
  the mutation_rate specification for EA methods in MethodSCOLIBEA

- Real mutationScale
  the mutation_scale specification for EA methods in MethodSCOLIBEA

- Real mutationMinScale
  the min_scale specification for mutation in EA methods in MethodSCOLIBEA

- Real initDelta
  the initial_delta specification for APPS/COBYLA/PS/SW methods in MethodAPPS, MethodSCOLIBCOB, MethodSCOLIBPS, and MethodSCOLIBSW

- Real threshDelta
  the threshold_delta specification for APPS/COBYLA/PS/SW methods in MethodAPPS, MethodSCOLIBCOB, MethodSCOLIBPS, and MethodSCOLIBSW

- Real contractFactor
  the contraction_factor specification for APPS/PS/SW methods in MethodAPPS, MethodSCOLIBPS, and MethodSCOLIBSW
• int newSolnsGenerated
  the new_solutions_generated specification for GA/EPSA methods in MethodSCOLIBEA

• int numberRetained
  the integer assignment to random, chc, or elitist in the replacement_type specification for GA/EPSA methods in MethodSCOLIBEA

• bool expansionFlag
  the no_expansion specification for APPS/PS/SW methods in MethodAPPS, MethodSCOLIBPS, and MethodSCOLIBSW

• int expandAfterSuccess
  the expand_after_success specification for PS/SW methods in MethodSCOLIBPS and MethodSCOLIBSW

• int contractAfterFail
  the contract_after_failure specification for the SW method in MethodSCOLIBSW

• int mutationRange
  the mutation_range specification for the pga_int method in MethodSCOLIBEA

• int totalPatternSize
  the total_pattern_size specification for PS methods in MethodSCOLIBPS

• bool randomizeOrderFlag
  the stochastic specification for the PS method in MethodSCOLIBPS

• String selectionPressure
  the fitness_type specification for EA methods in MethodSCOLIBEA

• String replacementType
  the replacement_type specification for EA methods in MethodSCOLIBEA

• String crossoverType
  the crossover_type specification for EA methods in MethodSCOLIBEA

• String mutationType
  the mutation_type specification for EA methods in MethodSCOLIBEA

• String exploratoryMoves
  the exploratory_moves specification for the PS method in MethodSCOLIBPS

• String patternBasis
  the pattern_basis specification for APPS/PS methods in MethodAPPS and MethodSCOLIBPS

• String betaSolverName
  beta solvers don’t need documentation

• String evalSynchronize
  the synchronization setting for parallel pattern search methods in MethodSCOLIBPS and MethodAPPS

• size_t numCrossPoints
  The number of crossover points or multi-point schemes.

• size_t numParents
  The number of parents to use in a crossover operation.

• size_t numOffspring
  The number of children to produce in a crossover operation.

• String fitnessType
  the fitness assessment operator to use.

• String convergenceType
The means by which this JEGA should converge.

- **Real percentChange**
  The minimum percent change before convergence for a fitness tracker converger.

- **size_t numGenerations**
  The number of generations over which a fitness tracker converger should track.

- **Real fitnessLimit**
  The cutoff value for survival in fitness limiting selectors (e.g., belowLimit selector).

- **Real shrinkagePercent**
  The minimum percentage of the requested number of selections that must take place on each call to the selector (0, 1).

- **String nichingType**
  The niching type.

- **RealVector nicheVector**
  The discretization percentage along each objective.

- **size_t numDesigns**
  The maximum number of designs to keep when using the maxDesigns nicher.

- **String postProcessorType**
  The post processor type.

- **RealVector distanceVector**
  The discretization percentage along each objective.

- **String initializationType**
  The means by which the JEGA should initialize the population.

- **String flatFile**
  The filename to use for initialization.

- **String logFile**
  The filename to use for logging.

- **int populationSize**
  The populationSize specification for GA methods in MethodSCOLIBEA

- **bool printPopFlag**
  The print_each_pop flag to set the printing of the population at each generation.

- **Real volBoxSize**
  The volume_boxsize_limit for the DIRECT method in MethodNCSUDC

- **int numSymbols**
  The symbols specification for DACE methods

- **bool mainEffectsFlag**
  The main_effects specification for sampling methods in MethodDDACE

- **bool latinizeFlag**
  The latinize specification for FSU QMC and CVT methods in MethodFSUDACE

- **bool volQualityFlag**
  The quality_metrics specification for sampling methods (FSU QMC and CVT methods in MethodFSUDACE)

- **IntVector sequenceStart**
  The sequenceStart specification in MethodFSUDACE

- **IntVector sequenceLeap**
the sequenceLeap specification in MethodFSUDACE

- IntVector primeBase
  the primeBase specification in MethodFSUDACE

- int numTrials
  the numTrials specification in MethodFSUDACE

- String trialType
  the trial_type specification in MethodFSUDACE

- int randomSeed
  the seed specification for COLINY, NonD, & DACE methods

- Real initMeshSize
  the initMeshSize choice for NOMAD in MethodNOMADDC

- Real minMeshSize
  the minMeshSize choice for NOMAD in MethodNOMADDC

- String historyFile
  the HISTORY_FILE specification for NOMAD

- String displayFormat
  the DISPLAY_STATS specification for NOMAD

- Real vns
  the VNS specification for NOMAD

- int neighborOrder
  the NEIGHBOR_ORDER specification for NOMAD

- bool showAllEval
  the DISPLAY_ALL_EVAL specification for NOMAD

- String useSurrogate
  the HAS_SGTE specification for NOMAD

- int numSamples
  the samples specification for NonD & DACE methods

- bool fixedSeedFlag
  flag for fixing the value of the seed among different NonD/DACE sample sets. This results in the use of the same
  sampling stencil/pattern throughout an execution with repeated sampling.

- bool fixedSequenceFlag
  flag for fixing the sequence for Halton or Hammersley QMC sample sets. This results in the use of the same
  sampling stencil/pattern throughout an execution with repeated sampling.

- bool vbdFlag
  the var_based_decomp specification for a variety of sampling methods

- Real vbdDropTolerance
  the var_based_decomp tolerance for omitting index output

- bool backfillFlag
  the backfill option allows one to augment in LHS sample by enforcing the addition of unique discrete variables
to the sample

- bool pcaFlag
  Flag to specify the calculation of principal components when using LHS.

- Real percentVarianceExplained
  The percentage of variance explained by using a truncated number of principal components in PCA.
- bool wilksFlag
  
  Flag to specify use of Wilks formula to calculate num samples.

- unsigned short wilksOrder
  
  Wilks order parameter.

- Real wilksConfidenceLevel
  
  Wilks confidence interval parameter.

- short wilksSidedInterval
  
  Wilks sided interval type.

- unsigned short vbdOrder
  
  A sub-specification of vbdFlag: interaction order limit for calculation/output of component VBD indices

- short covarianceControl
  
  Restrict the calculation of a full response covariance matrix for high dimensional outputs: \{DEFAULT,DIAGONAL,FULL\}_COVARIANCE

- String rngName
  
  The basic random-number generator for NonD

- short refinementType
  
  Refinement type for stochastic expansions from dimension refinement keyword group

- short refinementControl
  
  Refinement control for stochastic expansions from dimension refinement keyword group

- short nestingOverride
  
  Override for default point nesting policy: NO_NESTING_OVERRIDE, NESTED, or NON_NESTED

- short growthOverride
  
  Override for default point growth restriction policy: NO_GROWTH_OVERRIDE, RESTRICTED, or UNRESTRICTED

- short expansionType
  
  Enumeration for u-space type that defines u-space variable targets for probability space transformations: EXTENDED_U (default), ASKEY_U, STD_NORMAL_U, or STD_UNIFORM_U

- bool piecewiseBasis
  
  Boolean indicating presence of piecewise keyword

- short expansionBasisType
  
  Enumeration for type of basis in sparse grid interpolation (Pecos::\{NODAL,HIERARCHICAL\}_INTERPOLANT) or regression (Pecos::\{TENSOR_PRODUCT,TOTAL_ORDER,ADAPTED\}_BASIS).

- UShortArray quadratureOrderSeq
  
  The quadrature order sequence specification in MethodNonDPCE and MethodNonDSC

- UShortArray sparseGridLevelSeq
  
  The sparse grid level sequence specification in MethodNonDPCE and MethodNonDSC

- UShortArray expansionOrderSeq
  
  The expansion order sequence specification in MethodNonDPCE

- SizetArray collocationPointsSeq
  
  The collocation points sequence specification in MethodNonDPCE

- SizetArray expansionSamplesSeq
  
  The expansion samples sequence specification in MethodNonDPCE

- unsigned short quadratureOrder
  
  The quadrature order specification in MethodNonDPCE and MethodNonDSC
**unsigned short sparseGridLevel**

the sparse\_grid\_level specification in MethodNonDPCE and MethodNonDSC

**unsigned short expansionOrder**

the expansion\_order specification in MethodNonDPCE

**size\_t collocationPoints**

the collocation\_points specification in MethodNonDPCE

**size\_t expansionSamples**

the expansion\_samples specification in MethodNonDPCE

**String expansionSampleType**

allows for incremental PCE construction using the incremental\_lhs specification in MethodNonDPCE

**RealVector anisoDimPref**

the dimension\_preference specification for tensor and sparse grids and expansion orders in MethodNonDPCE and MethodNonDSC

**unsigned short cubIntOrder**

the cubature\_integrand specification in MethodNonDPCE

**Real collocationRatio**

the collocation\_ratio specification in MethodNonDPCE

**Real collocRatioTermsOrder**

order applied to the number of expansion terms when applying or computing the collocation ratio within regression PCE; based on the ratio\_order specification in MethodNonDPCE

**short regressionType**

type of regression: LS, OMP, BP, BPDN, LARS, or LASSO

**short lsRegressionType**

type of least squares regression: SVD or EQ\_CON\_QR

**RealVector regressionNoiseTol**

noise tolerance(s) for OMP, BPDN, LARS, and LASSO

**Real regressionL2Penalty**

L2 regression penalty for a variant of LASSO known as the elastic net method (default of 0 gives standard LASSO)

**bool crossValidation**

flag indicating the use of cross-validation across expansion orders (given a prescribed maximum order) and, for some methods, noise tolerances

**bool crossValidNoiseOnly**

flag indicating the restriction of cross-validation to estimate only the most effective noise tolerance; used to reduce cost from performing CV over both noise tolerances and expansion orders

**unsigned short adaptedBasisAdvancements**

initial grid level for the ADAPTED\_BASIS\_EXPANDING\_FRONT approach to defining the candidate basis for sparse recovery (compressed sensing)

**bool normalizedCoeffs**

flag indicating the output of PCE coefficients corresponding to normalized basis polynomials

**String pointReuse**

allows PCE construction to reuse points from previous sample sets or data import using the reuse\_points specification in MethodNonDPCE

**bool tensorGridFlag**
flag for usage of a sub-sampled set of tensor-product grid points within regression PCE; based on the tensor-grid specification in MethodNonDPCE

- **UShortArray tensorGridOrder**
  
  order of tensor-product grid points that are sub-sampled within orthogonal least interpolation PCE; based on the tensor_grid specification in MethodNonDPCE

- **short multilevDiscrepEmulation**
  
  type of discrepancy emulation in multilevel methods: distinct or recursive

- **String importExpansionFile**
  
  the import_expansion_file specification in MethodNonDPCE

- **String exportExpansionFile**
  
  the export_expansion_file specification in MethodNonDPCE

- **unsigned short sampleType**
  
  the sample_type specification in MethodNonDMC, MethodNonDPCE, and MethodNonDSC

- **bool dOptimal**
  
  whether to generate D-optimal designs

- **size_t numCandidateDesigns**
  
  number of candidate designs in D-optimal design selection

- **unsigned short reliabilitySearchType**
  
  the type of limit state search in MethodNonDLocalRel (x_taylor_mean, x_taylor_mpp, x_two_point, u_taylor_mean, u_taylor_mpp, u_two_point, or no_approx) or MethodNonDGlobalRel (x_gaussian_process or u_gaussian_process)

- **String reliabilityIntegration**
  
  the first_order or second_order integration selection in MethodNonDLocalRel

- **unsigned short integrationRefine**
  
  the import, adapt_import, or mm_adapt_import integration refinement selection in MethodNonDLocalRel, MethodNonDPCE, and MethodNonDSC

- **IntVector refineSamples**
  
  Sequence of refinement samples, e.g., the size of the batch (e.g. number of supplemental points added) to be added to the build points for an emulator at each iteration.

- **SizetArray pilotSamples**
  
  the pilot_samples selection in MethodMultilevelMC

- **Real multilevEstimatorRate**
  
  the estimator_rate selection in MethodMultilevelPCE

- **short finalMomentsType**
  
  the final_moments specification in MethodNonD

- **short distributionType**
  
  the distribution cumulative or complementary specification in MethodNonD

- **short responseLevelTarget**
  
  the compute probabilities, reliabilities, or gen_reliabilities specification in MethodNonD

- **short responseLevelTargetReduce**
  
  the system series or parallel specification in MethodNonD

- **RealVectorArray responseLevels**
  
  the response_levels specification in MethodNonD

- **RealVectorArray probabilityLevels**
the probability_levels specification in MethodNonD

- RealVectorArray reliabilityLevels
  the reliability_levels specification in MethodNonD

- RealVectorArray genReliabilityLevels
  the gen_reliability_levels specification in MethodNonD

- int chainSamples
  the number of MCMC chain samples

- int buildSamples
  the number of samples to construct an emulator, e.g., for Bayesian calibration methods

- int samplesOnEmulator
  number of samples to perform on emulator

- int emulatorOrder
  The total order to be used in construction of a VPS surrogate.

- short emulatorType
  the emulator specification in MethodNonDBayesCalib

- String mcmcType
  the mcmc type specification in MethodNonDBayesCalib

- bool standardizedSpace
  use of standardized probability spaces for MCMC within Bayesian inference

- bool adaptPosteriorRefine
  flag indicating adaptive refinement of the emulator in regions of high posterior probability

- bool logitTransform
  flag indicating user activation of logit transform option within QUESO

- bool gpmsaNormalize
  whether to apply GPMSA-internal normalization

- bool posteriorStatsKL
  flag indicating the calculation of KL divergence between prior and posterior in Bayesian methods

- bool posteriorStatsMutual
  flag indicating the calculation of mutual information between prior and posterior in Bayesian methods

- unsigned short preSolveMethod
  the method used for performing a pre-solve for the MAP point

- String proposalCovType
  the type of proposal covariance: user, derivatives, or prior

- int proposalCovUpdates
  number of updates of the proposal covariance from computing the misfit Hessian using residual values and derivatives

- String proposalCovInputType
  the format of proposal covariance input: diagonal or matrix

- RealVector proposalCovData
  raw list of real data for the proposal covariance

- String proposalCovFile
  file from which to read proposal covariance in diagonal or matrix format

- String quesoOptionsFilename
- **String** `fitnessMetricType`
  the fitness metric type specification in `MethodNonDAdaptive`

- **String** `batchSelectionType`
  the batch selection type specification in `MethodNonDAdaptive`

- **String** `lipschitzType`
  the Lipschitz type specification in `MethodNonDPOFDarts` (e.g. either local or global estimation)

- **unsigned short** `calibrateErrorMode`
  calibration mode for observation error multipliers (CALIBRATE_*)

- **RealVector** `hyperPriorAlphas`
  hyperparameters inverse gamma prior alphas

- **RealVector** `hyperPriorBetas`
  hyperparameters inverse gamma prior alphas

- **int** `burnInSamples`
  number of MCMC samples to discard from acceptance chain

- **int** `subSamplingPeriod`
  period or skip in post-processing the acceptance chain

- **bool** `calModelDiscrepancy`
  flag to calculate model discrepancy

- **size_t** `numPredConfigs`
  number of prediction configurations at which to calculate model discrepancy

- **RealVector** `predictionConfigList`
  list of prediction configurations at which to calculate model discrepancy

- **String** `importPredConfigs`
  whether to import prediction configurations at which to calculate model discrepancy

- **unsigned short** `importPredConfigFormat`
  tabular format for prediction configurations import file

- **String** `modelDiscrepancyType`
  type of model discrepancy emulation

- **short** `approxCorrectionOrder`
  correction order for either gaussian process or polynomial model discrepancy calculations: 0 (=constant), 1 (=linear), 2 (=quadratic)

- **String** `exportCorrModelFile`
  specify the name of file to which corrected model (model+discrepancy) calculations are output

- **unsigned short** `exportCorrModelFormat`
  tabular format for corrected model (model+discrepancy) export file

- **String** `exportCorrVarFile`
  specify the name of file to which corrected model variance calculations are output

- **unsigned short** `exportCorrVarFormat`
  tabular format for corrected model variance export file

- **String** `exportDiscrepFile`
  specify the name of file to which discrepancy calculations are output

- **unsigned short** `exportDiscrepFormat`
Chapter 13. Class Documentation

- bool adaptExpDesign
  Whether to perform adaptive Bayesian design of experiments
- String importCandPtsFile
  Whether to import candidate design points for adaptive Bayesian experimental design
- unsigned short importCandFormat
  Tabular format for the candidate design points import file
- size_t numCandidates
  Number of candidate designs for adaptive Bayesian experimental design
- int maxHifiEvals
  Maximum number of high fidelity model runs to be used for adaptive Bayesian experimental design
- bool mutualInfoKSG2
  Indicate that the KSG2 algorithm is to be employed in the calculation of the mutual information
- int numChains
  Number of concurrent chains
- int numCR
  Number of CR-factors
- int crossoverChainPairs
  Number of crossover chain pairs
- Real grThreshold
  Threshold for the Gelman-Rubin statistic
- int jumpStep
  How often to perform a long jump in generations
- int numPushforwardSamples
  Number of samples from the prior that is pushed forward through the model to obtain the initial set of pushforward samples.
- String dataDistType
  The type of data distribution: kde, or gaussian
- String dataDistCovInputType
  The format of data distribution gaussian covariance input: diagonal or matrix
- RealVector dataDistMeans
  Raw list of real data for the data distribution gaussian means
- RealVector dataDistCovariance
  Raw list of real data for the data distribution gaussian covariance
- String dataDistFile
  File from which to read data distribution data (covariance or samples)
- String posteriorDensityExportFilename
  The filename of the export file containing an arbitrary set of samples and their corresponding density values.
- String posteriorSamplesExportFilename
  The filename of the export file containing samples from the posterior and their corresponding density values.
- String posteriorSamplesImportFilename
  The filename of the import file containing samples at which the posterior will be evaluated.
- bool generatePosteriorSamples
Flag specifying whether to generate random samples from the posterior.

- **bool evaluatePosteriorDensity**
  
  Flag specifying whether to evaluate the posterior density at a set of samples.

- **RealVector finalPoint**
  
  *the final_point specification in MethodPSVPS*

- **RealVector stepVector**
  
  *the step_vector specification in MethodPSVPS and MethodPSCPS*

- **int numSteps**
  
  *the num_steps specification in MethodPSVPS*

- **IntVector stepsPerVariable**
  
  *the deltas_per_variable specification in MethodPSCPS*

- **RealVector listOfPoints**
  
  *the list_of_points specification in MethodPSLPS*

- **String pstudyFilename**
  
  *the import_points_file spec for a file-based parameter study*

- **unsigned short pstudyFileFormat**
  
  *tabular format for the parameter study points file*

- **bool pstudyFileActive**
  
  *whether to import active variables only*

- **UShortArray varPartitions**
  
  *the partitions specification for PStudy method in MethodPSMPS*

- **Real refinementRate**
  
  *rate of mesh refinement in Richardson extrapolation*

- **String importBuildPtsFile**
  
  *the file name from the import_build_points_file specification*

- **unsigned short importBuildFormat**
  
  *tabular format for the build point import file*

- **bool importBuildActive**
  
  *whether to import active variables only*

- **String importApproxPtsFile**
  
  *the file name from the import_approx_points_file specification*

- **unsigned short importApproxFormat**
  
  *tabular format for the approx point import file*

- **bool importApproxActive**
  
  *whether to import active variables only*

- **String exportApproxPtsFile**
  
  *the file name from the export_approx_points_file specification*

- **unsigned short exportApproxFormat**
  
  *tabular format for the approx point export file*

- **String exportMCMCPtsFile**
  
  *the file name from the export_mcmc_points_file specification*

- **bool exportSampleSeqFlag**
  
  *flag for exporting the sequence of sample increments within multilevel sampling from the export_sample_sequence specification*

- **unsigned short exportSamplesFormat**
  
  *tabular format for the MCMC chain and MLMC sample sequence exports*
Private Member Functions

- **DataMethodRep ()**
  constructor
- **~DataMethodRep ()**
  destructor
- void **write (std::ostream &s) const**
  write a DataInterfaceRep object to an std::ostream
- void **read (MPIUnpackBuffer &s)**
  read a DataInterfaceRep object from a packed MPI buffer
- void **write (MPIPackBuffer &s) const**
  write a DataInterfaceRep object to a packed MPI buffer

Private Attributes

- int **referenceCount**
  number of handle objects sharing this DataMethodRep

Friends

- class **DataMethod**
  the handle class can access attributes of the body class directly

13.32.1 Detailed Description

Body class for method specification data.

The DataMethodRep class is used to contain the data from a method keyword specification. Default values are managed in the DataMethodRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataMethodList is private.

The documentation for this class was generated from the following files:

- DataMethod.hpp
- DataMethod.cpp

13.33 DataModel Class Reference

Handle class for model specification data.

Public Member Functions

- **DataModel ()**
  constructor
- **DataModel (const DataModel &)**
  copy constructor
- **~DataModel ()**
  destructor
- **DataModel & operator= (const DataModel &)**
assignment operator

- void write (std::ostream &s) const
  write a DataModel object to an std::ostream
- void read (MPIUnpackBuffer &s)
  read a DataModel object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  write a DataModel object to a packed MPI buffer
- DataModelRep * data_rep ()
  return dataModelRep

**Static Public Member Functions**

- static bool id_compare (const DataModel &dm, const std::string &id)
  compares the idModel attribute of DataModel objects

**Private Attributes**

- DataModelRep * dataModelRep
  pointer to the body (handle-body idiom)

**Friends**

- class ProblemDescDB
- class NIDRProblemDescDB

### 13.33.1 Detailed Description

Handle class for model specification data.

The DataModel class is used to provide a memory management handle for the data in DataModelRep. It is populated by IDRProblemDescDB::model_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataModel objects is maintained in ProblemDescDB::dataModelList, one for each model specification in an input file.

The documentation for this class was generated from the following files:

- DataModel.hpp
- DataModel.cpp

### 13.34 DataModelRep Class Reference

Body class for model specification data.

**Public Attributes**

- String idModel
  string identifier for the model specification data set (from the id_model specification in ModelIndControl)
- String modelType
  model type selection: single, surrogate, or nested (from the model type specification in ModelIndControl)
• String variablesPointer
  string pointer to the variables specification to be used by this model (from the variables_pointer specification in ModelIndControl)

• String interfacePointer
  string pointer to the interface specification to be used by this model (from the interface_pointer specification in ModelSingle and the optional_interface_pointer specification in ModelNested)

• String responsesPointer
  string pointer to the responses specification to be used by this model (from the responses_pointer specification in ModelIndControl)

• bool hierarchicalTags
  whether this model and its children will add hierarchy-based tags to eval ids

• String subMethodPointer
  pointer to a sub-iterator used for global approximations (from the dace_method_pointer specification in ModelSurrG) or by nested models (from the sub_method_pointer specification in ModelNested)

• String solutionLevelControl
  (state) variable identifier that defines a set or range of solution level controls (space/time discretization levels, iterative convergence tolerances, etc.) for defining a secondary hierarchy of fidelity within the scope of a single model form (from solution_level_control specification; see also ordered_model_fidelities)

• RealVector solutionLevelCost
  array of relative simulation costs corresponding to each of the solution levels (from solution_level_cost specification; see also solution_level_control); a scalar input is interpreted as a constant cost multiplier to be applied recursively

• IntSet surrogateFnIndices
  array specifying the response function set that is approximated

• String surrogateType
  the selected surrogate type: local_taylor, multipoint_tana, global_neural_network,mars,orthogonal,polynomial,gaussian, polynomial,kriging), or hierarchical

• String actualModelPointer
  pointer to the model specification for constructing the truth model used in constructing surrogates (from the actual_model_pointer specification in ModelSurrL and ModelSurrMP)

• StringArray orderedModelPointers
  an ordered list of model pointers (low to high) corresponding to a hierarchy of modeling fidelity (from the ordered_model_fidelities specification in ModelSurrH)

• int pointsTotal
  user-specified lower bound on total points with which to build the model (if reuse_points < pointsTotal, new samples will make up the difference)

• short pointsManagement
  points management configuration for DataFitSurrModel: DEFAULT_POINTS, MINIMUM_POINTS, or RECOMMENDED_POINTS

• String approxPointReuse
  sample reuse selection for building global approximations: none, all, region, or file (from the reuse_samples specification in ModelSurrG)

• String importBuildPtsFile
  the file name from the import_build_points_file specification in ModelSurrG

• unsigned short importBuildFormat
  tabular format for the build point import file
- `bool importBuildActive` 
  whether to import active variables only
- `String exportApproxPtsFile` 
  the file name from the export_approx_points_file specification in ModelSurrG
- `unsigned short exportApproxFormat` 
  tabular format for the approx point export file
- `bool exportSurrogate` 
  Option to turn on surrogate model export (export_model)
- `String modelExportPrefix` 
  the filename prefix for export_model
- `unsigned short modelExportFormat` 
  Format selection for export_model.
- `short approxCorrectionType` 
  correction type for global and hierarchical approximations: NO_CORRECTION, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, or COMBINED_CORRECTION (from the correction specification in ModelSurrG and ModelSurrH)
- `short approxCorrectionOrder` 
  correction order for global and hierarchical approximations: 0, 1, or 2 (from the correction specification in ModelSurrG and ModelSurrH)
- `bool modelUseDerivsFlag` 
  flags the use of derivatives in building global approximations (from the use_derivatives specification in ModelSurrG)
- `short polynomialOrder` 
  scalar integer indicating the order of the polynomial approximation (1=linear, 2=quadratic, 3=cubic; from the polynomial specification in ModelSurrG)
- `RealVector krigingCorrelations` 
  vector of correlations used in building a kriging approximation (from the correlations specification in ModelSurrG)
- `String krigingOptMethod` 
  optimization method to use in finding optimal correlation parameters: none, sampling, local, global
- `short krigingMaxTrials` 
  maximum number of trials in optimization of kriging correlations
- `RealVector krigingMaxCorrelations` 
  upper bound on kriging correlation vector
- `RealVector krigingMinCorrelations` 
  lower bound on kriging correlation vector
- `Real krigingNugget` 
  nugget value for kriging
- `short krigingFindNugget` 
  option to have Kriging find the best nugget value to use
- `short mlsWeightFunction` 
  weight function for moving least squares approximation
- `short rbfBases` 
  bases for radial basis function approximation
• short rbfMaxPts
  maximum number of points for radial basis function approximation
• short rbfMaxSubsets
  maximum number of subsets for radial basis function approximation
• short rbfMinPartition
  minimum partition for radial basis function approximation
• short marsMaxBases
  maximum number of bases for MARS approximation
• String marsInterpolation
  interpolation type for MARS approximation
• short annRandomWeight
  random weight for artificial neural network approximation
• short annNodes
  number of nodes for artificial neural network approximation
• Real annRange
  range for artificial neural network approximation
• bool domainDecomp
  whether domain decomposition is enabled
• String decompCellType
  type of local cell of domain decomp
• int decompSupportLayers
  number of support layers for each local basis function
• bool decompDiscontDetect
  whether discontinuity detection is enabled
• Real discontJumpThresh
  function value (jump) threshold for discontinuity detection in domain decomp
• Real discontGradThresh
  gradient threshold for discontinuity detection in domain decomp
• String trendOrder
  scalar integer indicating the order of the Gaussian process mean (0= constant, 1=linear, 2=quadratic, 3=cubic); from the gaussian process specification in ModelSurrG
• bool pointSelection
  flag indicating the use of point selection in the Gaussian process
• StringArray diagMetrics
  List of diagnostic metrics the user requests to assess the goodness of fit for a surrogate model.
• bool crossValidateFlag
  flag indicating the use of cross validation on the metrics specified
• int numFolds
  number of folds to perform in cross validation
• Real percentFold
  percentage of data to withhold for cross validation process
• bool pressFlag
  flag indicating the use of PRESS on the metrics specified
- String `importChallengePtsFile`  
  *the file name from the `challenge_points_file` specification in ModelSurrG*
- unsigned short `importChallengeFormat`  
  *tabular format of the challenge data file*
- bool `importChallengeActive`  
  *whether to import active variables only*
- String `optionalInterfRespPointer`  
  *string pointer to the responses specification used by the optional interface in nested models (from the `optional_interface_responses_pointer` specification in ModelNested)*
- StringArray `primaryVarMaps`  
  *the primary variable mappings used in nested models for identifying the lower level variable targets for inserting top level variable values (from the `primary_variable_mapping` specification in ModelNested)*
- StringArray `secondaryVarMaps`  
  *the secondary variable mappings used in nested models for identifying the (distribution) parameter targets within the lower level variables for inserting top level variable values (from the `secondary_variable_mapping` specification in ModelNested)*
- RealVector `primaryRespCoeffs`  
  *the primary response mapping matrix used in nested models for weighting contributions from the sub-iterator responses in the top level (objective) functions (from the `primary_response_mapping` specification in ModelNested)*
- RealVector `secondaryRespCoeffs`  
  *the secondary response mapping matrix used in nested models for weighting contributions from the sub-iterator responses in the top level (constraint) functions (from the `secondary_response_mapping` specification in ModelNested)*
- int `subMethodServers`  
  *number of servers for concurrent sub-iterator parallelism*
- int `subMethodProcs`  
  *number of processors for each concurrent sub-iterator partition*
- short `subMethodScheduling`  
  *scheduling approach for concurrent sub-iterator parallelism: {DEFAULT, MASTER, PEER}_SCHEDULING*
- int `initialSamples`  
  *initial samples to build the subspace model*
- unsigned short `subspaceSampleType`  
  *sampling method for building the subspace model*
- IntVector `refineSamples`  
  *refinement samples to add in each batch*
- int `maxIterations`  
  *maximum number of subspace build iterations*
- Real `convergenceTolerance`  
  *convergence tolerance on build process*
- bool `subspaceIdBingLi`  
  *Flag to use Bing Li method to identify active subspace dimension.*
- bool `subspaceIdConstantine`  
  *Flag to use Constantine method to identify active subspace dimension.*
- bool `subspaceIdEnergy`
Flag to use eigenvalue energy method to identify active subspace dimension.

- `bool subspaceBuildSurrogate`
  Flag to build surrogate over active subspace.
- `int subspaceDimension`
  Size of subspace.
- `unsigned short subspaceNormalization`
  Normalization to use when forming a subspace with multiple response functions.
- `int numReplicates`
  Number of bootstrap samples for subspace identification.
- `bool subspaceIdCV`
  Flag to use cross validation to identify active subspace dimension.
- `Real relTolerance`
  Relative tolerance used by cross validation subspace dimension id method
- `Real decreaseTolerance`
  Decrease tolerance used by cross validation subspace dimension id method
- `int subspaceCVMaxRank`
  Maximum rank considered by cross validation subspace dimension id method
- `bool subspaceCVIncremental`
  Flag to use incremental dimension estimation in the cross validation metric
- `unsigned short subspaceIdCVMethod`
  Contains which cutoff method to use in the cross validation metric.
- `bool autoRefine`
  Whether automatic surrogate refinement is enabled
- `int maxFunctionEvals`
  Maximum evals in refinement
- `String refineCVMetric`
  Metric to use in cross-validation guided refinement
- `int softConvergenceLimit`
  Max number of iterations in refinement without improvement
- `int refineCVFolds`
  Number of cross-validation folds in guided refinement
- `unsigned short adaptedBasisSparseGridLev`
  Sparse grid level for low-order PCE used to compute rotation matrix within adapted basis approach to dimension reduction
- `unsigned short adaptedBasisExpOrder`
  Expansion order for low-order PCE used to compute rotation matrix within adapted basis approach to dimension reduction
- `Real adaptedBasisCollocRatio`
  Collocation ratio for low-order PCE used to compute rotation matrix within adapted basis approach to dimension reduction
- `unsigned short randomFieldIdForm`
  Contains which type of random field model.
- `unsigned short analyticCovIdForm`
  Contains which type of analytic covariance function.
• Real **truncationTolerance**
  
  truncation tolerance on build process: percent variance explained

• String **propagationModelPointer**
  
  pointer to the model through which to propagate the random field

• String **rfDataFileName**
  
  File from which to build the random field.

**Private Member Functions**

• **DataModelRep ()**
  
  *constructor*

• **~DataModelRep ()**
  
  *destructor*

• void **write (std::ostream &s) const**
  
  write a DataModelRep object to an std::ostream

• void **read (MPIUnpackBuffer &s)**
  
  read a DataModelRep object from a packed MPI buffer

• void **write (MPIPackBuffer &s) const**
  
  write a DataModelRep object to a packed MPI buffer

**Private Attributes**

• int **referenceCount**
  
  number of handle objects sharing this DataModelRep

**Friends**

• class **DataModel**
  
  the handle class can access attributes of the body class directly

### 13.34.1 Detailed Description

Body class for model specification data.

The DataModelRep class is used to contain the data from a model keyword specification. Default values are managed in the DataModelRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataModelList is private.

The documentation for this class was generated from the following files:

• DataModel.hpp
• DataModel.cpp

### 13.35 DataResponses Class Reference

Handle class for responses specification data.
Public Member Functions

- **DataResponses ()**
  
  constructor
- **DataResponses (const DataResponses &)**
  
  copy constructor
- **~DataResponses ()**
  
  destructor
- **DataResponses & operator= (const DataResponses &)**
  
  assignment operator
- **void write (std::ostream &s) const**
  
  write a DataResponses object to an std::ostream
- **void read (MPIUnpackBuffer &s)**
  
  read a DataResponses object from a packed MPI buffer
- **void write (MPIPackBuffer &s) const**
  
  write a DataResponses object to a packed MPI buffer
- **DataResponsesRep * data_rep ()**
  
  return dataRespRep

Static Public Member Functions

- **static bool id_compare (const DataResponses &dr, const std::string &id)**
  
  compares the idResponses attribute of DataResponses objects

Private Attributes

- **DataResponsesRep * dataRespRep**
  
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

13.35.1 Detailed Description

Handle class for responses specification data.

The DataResponses class is used to provide a memory management handle for the data in DataResponsesRep. It is populated by IDRProblemDescDB::responses_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataResponses objects is maintained in ProblemDescDB::dataResponsesList, one for each responses specification in an input file.

The documentation for this class was generated from the following files:

- DataResponses.hpp
- DataResponses.cpp
13.36  DataResponsesRep Class Reference

Body class for responses specification data.

Public Attributes

- **String idResponses**
  - string identifier for the responses specification data set (from the id_responses specification in RespSetId)
- **CharArray responseLabels**
  - the response labels array (from the response_descriptors specification in RespLabels)
- **size_t numObjectiveFunctions**
  - number of objective functions (from the num_objective_functions specification in RespFnOpt)
- **size_t numLeastSqTerms**
  - number of least squares terms (from the num_calibration_terms specification in RespFnLS)
- **size_t numNonlinearIneqConstraints**
  - number of nonlinear inequality constraints (from the num_nonlinear_inequality_constraints specification in RespFnOpt)
- **size_t numNonlinearEqConstraints**
  - number of nonlinear equality constraints (from the num_nonlinear_equality_constraints specification in RespFnOpt)
- **size_t numResponseFunctions**
  - number of generic response functions (from the num_response_functions specification in RespFnGen)
- **size_t numScalarObjectiveFunctions**
  - scalar_objectives: number of objective functions which are scalar
- **size_t numScalarLeastSqTerms**
  - scalar_calibration_terms: number of calibration terms which are scalar
- **size_t numScalarNonlinearIneqConstraints**
  - number of scalar nonlinear inequality constraints (from the num_scalar_nonlinear_inequality_constraints specification in RespFnOpt)
- **size_t numScalarNonlinearEqConstraints**
  - number of scalar nonlinear equality constraints (from the num_scalar_nonlinear_equality_constraints specification in RespFnOpt)
- **size_t numScalarResponseFunctions**
  - scalar_responses: number of response functions which are scalar
- **size_t numFieldObjectiveFunctions**
  - field_objectives: number of objective functions which are field-valued
- **size_t numFieldLeastSqTerms**
  - field_calibration_terms: number of calibration terms which are field-valued
- **size_t numFieldNonlinearIneqConstraints**
  - number of field nonlinear inequality constraints (from the num_field_nonlinear_inequality_constraints specification in RespFnOpt)
- **size_t numFieldNonlinearEqConstraints**
  - number of field nonlinear equality constraints (from the num_field_nonlinear_equality_constraints specification in RespFnOpt)
- **size_t numFieldResponseFunctions**
CHAPTER 13. CLASS DOCUMENTATION

- **field_responses**: number of response functions which are field-valued
  - **StringArray** `primaryRespFnSense`
    - optimization sense for each objective function: minimize or maximize
  - **RealVector** `primaryRespFnWeights`
    - vector of weightings for multiobjective optimization or weighted nonlinear least squares (from the multi_objective_weights specification in RespFnOpt and the least_squares_weights specification in RespFnLS)
  - **RealVector** `nonlinearIneqLowerBnds`
    - vector of nonlinear inequality constraint lower bounds (from the nonlinear_inequality_lower_bounds specification in RespFnOpt)
  - **RealVector** `nonlinearIneqUpperBnds`
    - vector of nonlinear inequality constraint upper bounds (from the nonlinear_inequality_upper_bounds specification in RespFnOpt)
  - **RealVector** `nonlinearEqTargets`
    - vector of nonlinear equality constraint targets (from the nonlinear_equality_targets specification in RespFnOpt)
  - **StringArray** `primaryRespFnScaleTypes`
    - vector of primary response function scaling types (from the objective_function_scale_types specification in RespFnOpt and the least_squares_term_scale_types specification in RespFnLS)
  - **RealVector** `primaryRespFnScales`
    - vector of primary response function scaling factors (from the objective_function_scales specification in RespFnOpt and the least_squares_term_scales specification in RespFnLS)
  - **StringArray** `nonlinearIneqScaleTypes`
    - vector of nonlinear inequality constraint scaling types (from the nonlinear_inequality_scale_types specification in RespFnOpt)
  - **RealVector** `nonlinearIneqScales`
    - vector of nonlinear inequality constraint scaling factors (from the nonlinear_inequality_scales specification in RespFnOpt)
  - **StringArray** `nonlinearEqScaleTypes`
    - vector of nonlinear equality constraint scaling types (from the nonlinear_equality_scale_types specification in RespFnOpt)
  - **RealVector** `nonlinearEqScales`
    - vector of nonlinear equality constraint scaling factors (from the nonlinear_equality_scales specification in RespFnOpt)
  - **bool** `calibrationDataFlag`
    - whether calibration data was specified
  - **size_t** `numExperiments`
    - number of distinct experiments in experimental data
  - **size_t** `numExpConfigVars`
    - number of experimental configuration vars (state variables) in each row of data
  - **RealVector** `expConfigVars`
    - list of num_experiments x num_config_vars configuration variable values
  - **RealVector** `simVariance`
    - list of variances of errors to be added to simulation responses
  - **bool** `interpolateFlag`
whether one should interpolate between the experiment and simulation field data

- **RealVector expObservations**
  - list of_num_calibration_terms observation data
- **RealVector expStdDeviations**
  - list of 1 or_num_calibration_terms observation standard deviations
- **String scalarDataFileName**
  - name of experimental data file containing response data (with optional state variable and sigma data) to read
- **unsigned short scalarDataFormat**
  - tabular format of the scalar data file
- **String gradientType**
  - gradient type: none, numerical, analytic, or mixed (from the no_gradients, numerical_gradients, analytic_gradients, and mixed_gradients specifications in RespGrad)
- **String hessianType**
  - Hessian type: none, numerical, quasi, analytic, or mixed (from the no_hessians, numerical_hessians, quasi_hessians, analytic_hessians, and mixed_hessians specifications in RespHess)
- **bool ignoreBounds**
  - option to ignore bounds when doing finite differences (default is to honor bounds)
- **bool centralHess**
  - Temporary(?) option to use old 2nd-order diffs when computing finite-difference Hessians; default is forward differences.
- **String quasiHessianType**
  - quasi-Hessian type: bfgs, damped_bfgs, or sr1 (from the bfgs and sr1 specifications in RespHess)
- **String methodSource**
  - numerical gradient method source: dakota or vendor (from the method_source specification in RespGradNum and RespGradMixed)
- **String intervalType**
  - numerical gradient interval type: forward or central (from the interval_type specification in RespGradNum and RespGradMixed)
- **RealVector fdGradStepSize**
  - vector of finite difference step sizes for numerical gradients, one step size per active continuous variable, used in computing 1st-order forward or central differences (from the fd_gradient_step_size specification in RespGradNum and RespGradMixed)
- **String fdGradStepType**
  - type of finite difference step to use for numerical gradient: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x
- **RealVector fdHessStepSize**
  - vector of finite difference step sizes for numerical Hessians, one step size per active continuous variable, used in computing 1st-order gradient-based differences and 2nd-order function-based differences (from the fd_hessian_step_size specification in RespHessNum and RespHessMixed)
- **String fdHessStepType**
  - type of finite difference step to use for numerical Hessian: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x
- **IntSet idNumericalGrads**
  - mixed gradient numerical identifiers (from the id_numerical_gradients specification in RespGradMixed)
- **IntSet idAnalyticGrads**
mixed gradient analytic identifiers (from the \texttt{id\_analytic\_gradients} specification in \texttt{RespGradMixed})

- \texttt{IntSet idNumericalHessians}
  
  mixed Hessian numerical identifiers (from the \texttt{id\_numerical\_hessians} specification in \texttt{RespHessMixed})

- \texttt{IntSet idQuasiHessians}
  
  mixed Hessian quasi identifiers (from the \texttt{id\_quasi\_hessians} specification in \texttt{RespHessMixed})

- \texttt{IntSet idAnalyticHessians}
  
  mixed Hessian analytic identifiers (from the \texttt{id\_analytic\_hessians} specification in \texttt{RespHessMixed})

- \texttt{IntVector fieldLengths}

  number of entries in each field

- \texttt{IntVector numCoordsPerField}

  number of coordinates per field

- \texttt{bool readFieldCoords}

  Field data related storage: whether to read simulation field coordinates.

- \texttt{StringArray varianceType}

  Array which specifies the sigma type per response (none, one constant value, one per response (vector) or a full covariance matrix

---

**Private Member Functions**

- \texttt{DataResponsesRep ()}

  constructor

- \texttt{\~{DataResponsesRep} ()}

  destructor

- \texttt{void write (std::ostream &s) const}

  write a \texttt{DataResponsesRep} object to an std::ostream

- \texttt{void read (MPIUnpackBuffer &s)}

  read a \texttt{DataResponsesRep} object from a packed MPI buffer

- \texttt{void write (MPIPackBuffer &s) const}

  write a \texttt{DataResponsesRep} object to a packed MPI buffer

---

**Private Attributes**

- \texttt{int referenceCount}

  number of handle objects sharing this dataResponsesRep

---

**Friends**

- \texttt{class DataResponses}

  the handle class can access attributes of the body class directly
13.36.1 Detailed Description

Body class for responses specification data.

The DataResponsesRep class is used to contain the data from a responses keyword specification. Default values are managed in the DataResponsesRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataResponsesList is private.

The documentation for this class was generated from the following files:

- DataResponses.hpp
- DataResponses.cpp

13.37 DataTransformModel Class Reference

Data transformation specialization of RecastModel.

Inheritance diagram for DataTransformModel:

```
Model
  RecastModel
  DataTransformModel
```

Public Member Functions

- **DataTransformModel** (const Model &sub_model, const ExperimentData &exp_data, size_t num_hyper=0, unsigned short mult_mode=CALIBRATE_NONE, short recast_resp_deriv_order=1)
  
  *standard constructor*

- **~DataTransformModel** ()
  
  *destructor*

- void **data_transform_response** (const Variables &sub_model_vars, const Response &sub_model_resp, Response &residual_resp)
  
  *Convenience function to help recover a residual response from the submodel.*

- void **data_resize** ()
  
  *The size of the ExperimentData changed; update the residualModel size.*

- void **print_best_responses** (std::ostream &, const Variables &best_submodel_vars, const Response &best_submodel_resp, size_t num_best, size_t best_ind)
  
  *manage best responses including residuals and model responses per config*

Protected Types

- typedef std::map<int, IntResponseMap> IntIntResponseMapMap
Protected Member Functions

- void `gen_primary_resp_map` (const `SharedResponseData` &srd, `Sizet2DArray` &primary_resp_map_indices, `BoolDequeArray` &nonlinear_resp_map) const
  
  compute the primary response map for a data transform `RecastModel`

- void `derived_evaluate` (const `ActiveSet` &set)
  
  specialization of evaluate that iterates over configuration variables

- void `derived_evaluate_nowait` (const `ActiveSet` &set)
  
  specialization of evaluate that iterates over configuration variables

- const `IntResponseMap` & `derived_synchronize` ()
  
  synchronize all evaluations (all residuals for all experiment configurations)

- const `IntResponseMap` & `derived_synchronize_nowait` ()
  
  return any evaluations for which all experiment configurations have completed

- const `IntResponseMap` & `filter_submodel_responses` ()

- void `cache_submodel_responses` (const `IntResponseMap` &smRespMap, bool deep_copy)
  
  cache the subModel responses into a per-RecastModel eval ID map

- void `collect_residuals` (bool collect_all)
  
  collect any (or force all) completed subModel evals and populate recastResponseMap with residuals for those that
  are fully completed

- void `transform_response_map` (const `IntResponseMap` &submodelResp, const `Variables` &recast_vars, `Response` &residual_resp)
  
  transform a set of per-configuration subModel Responses to a single evaluation's residuals

- void `scale_response` (const `Variables` &submodel_vars, const `Variables` &recast_vars, `Response` &recast_response)
  
  scale the populated residual response by any covariance information, including hyper-parameter multipliers

- void `init_continuous_vars` ()
  
  Initialize continuous variable values/labels.

- template<typename T>
  
  void `expand_array` (const `SharedResponseData` &srd, const T &submodel_array, size_t recast_size, T &recast_array) const
  
  (if non-empty) expand submodel_array by replicates to populate a recast_array

- template<typename T>
  
  void `expand_scales_array` (const `SharedResponseData` &srd, const T &submodel_array, size_t submodel_size, size_t recast_size, T &recast_array) const
  
  If size greater than 1, expand submodel_array by replicates to populate a pre-sized recast_array, otherwise copy.

- void `print_residual_response` (const `Response` &resid_resp)

- void `recover_submodel_responses` (std::ostream &s, const `Variables` &best_submodel_vars, size_t num_best, size_t best_ind, `Response` &residual_resp)

Static Protected Member Functions

- static `SizetArray` `variables_expand` (const `Model` &sub_model, size_t num_hyper)

  expand the variable counts to account for hyper-parameters

- static int `get_hyperparam_vc_index` (const `Model` &sub_model)

  determine the index into vc_totals corresponding to where the hyper-parameters go

- static short `response_order` (const `Model` &sub_model, short recast_resp_order=1)
helper to compute the recast response order during member initialization; recast respOrder passed is the minimum request client needs

- static void vars_mapping (const Variables &recast_vars, Variables &submodel_vars)
  
  map the inbound expanded variables to the sub-model, discarding hyperparams (assumes hyper-parameters are at end of active continuous variables)

- static void set_mapping (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  
  map the inbound ActiveSet to the sub-model (map derivative variables)

- static void primary_resp_differencer (const Variables &submodel_vars, const Variables &recast_vars, const Response &submodel_response, Response &recast_response)
  
  Recast callback function to difference residuals with observed data.

Protected Attributes

- const ExperimentData & expData
  
  Reference to the experiment data used to construct this Model.

- size_t numHyperparams
  
  Number of calibrated variance multipliers.

- unsigned short obsErrorMultiplierMode
  
  Calibration mode for the hyper-parameters.

- IntIntResponseMapMap cachedResp

Static Protected Attributes

- static DataTransformModel * dtModelInstance
  
  static pointer to this class for use in static callbacks

Additional Inherited Members

13.37.1 Detailed Description

Data transformation specialization of RecastModel.

Specialization of RecastModel to create a residual model that maps (1) from an augmented set of calibration parameters (including hyper-parameters) to those needed by the underlying simulation model and (2) from the simulation model response to a set of residuals, whose overall size may differ from the simulation (sub-model) response. The residuals may be scaled by experiment covariance information. This class provides a simple constructor that forwards to the more complicated RecastModel API

13.37.2 Constructor & Destructor Documentation

DataTransformModel ( const Model & sub_model, const ExperimentData & exp_data, size_t num Hyper = 0, unsigned short multi_mode = CALIBRATE_NONE, short recast_resp_deriv_order = 1 )

standard constructor

This constructor computes various indices and mappings, then updates the properties of the RecastModel. Hyper-parameters are assumed to trail the active continuous variables when presented to this RecastModel
References Dakota::abort_handler(), Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), DataTransformModel::dtModelInstance, DataTransformModel::expand_array(), DataTransformModel::expand_scales_array(), DataTransformModel::expData, DataTransformModel::gen_primary_resp_map(), Model::icv(), Model::idiv(), Model::idrv(), Model::idsv(), Model::inactive_view(), DataTransformModel::init_continuous_vars(), RecastModel::init_maps(), ExperimentData::num_config_vars(), Model::num_primary_fns(), Model::num_secondary_fns(), ExperimentData::num_total_exppoints(), DataTransformModel::numHyperparams, DataTransformModel::primary_resp_differencer(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primaryRespFnSense, Model::primaryRespFnWts, Model::scaling_options(), Model::scalingOptions, DataTransformModel::set_mapping(), Response::shared_data(), RecastModel::subModel, and DataTransformModel::vars_mapping().

13.37.3 Member Function Documentation

SizetArray variables_expand ( const Model & sub_model, size_t num_hyper ) [static], [protected]
expand the variable counts to account for hyper-parameters
Incorporate the hyper parameters into Variables, assuming they are at the end of the active continuous variables. For example, append them to continuous design or continuous aleatory uncertain.
References SharedVariablesData::components_totals(), Model::current_variables(), DataTransformModel::get_hyperparam_vc_index(), Variables::shared_data(), and Dakota::svd().

void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]
specialization of evaluate that iterates over configuration variables
Blocking evaluation over all experiment configurations to compute a single set of expanded residuals. If the subModel supports asynchronous evaluate_nowait(), do the configuration evals concurrently and then synchronize.
Reimplemented from Model.
References Response::active_set(), Model::asynch_flag(), ExperimentData::config_vars(), Model::current_response(), Model::currentResponse, Model::currentVariables, RecastModel::derived_evaluate(), Model::evaluate(), Model::evaluate_nowait(), Model::evaluation_id(), DataTransformModel::expData, DataTransformModel::filter_submodel_responses(), ExperimentData::form_residuals(), Model::inactive_variables(), ExperimentData::num_experiments(), Model::outputLevel, RecastModel::recastIdMap, RecastModel::recastModelEvalCntr, DataTransformModel::scale_response(), RecastModel::subModel, DataTransformModel::transform_response_map(), RecastModel::transform_set(), and RecastModel::transform_variables().

void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]
specialization of evaluate that iterates over configuration variables
Non-blocking evaluation (scheduling) over all experiment configurations. Assumes that if this model supports nowait, its subModel does too and schedules them all.
Reimplemented from Model.
References ExperimentData::config_vars(), Variables::copy(), Model::current_variables(), Model::currentVariables, RecastModel::derived_evaluate_nowait(), Model::evaluate_nowait(), Model::evaluation_id(), DataTransformModel::expData, Model::inactive_variables(), ExperimentData::num_experiments(), Model::outputLevel, RecastModel::recastIdMap, RecastModel::recastModelEvalCntr, RecastModel::recastSetMap, RecastModel::recastVarsMap, RecastModel::subModel, RecastModel::transform_set(), and RecastModel::transform_variables().

const IntResponseMap & derived_synchronize ( ) [protected], [virtual]
synchronize all evaluations (all residuals for all experiment configurations)
Collect all the subModel evals and build the residual sets for all evaluations. Like rekey functions in DakotaModel, but many sub-model to one recast-model. For the blocking synchronize case, we force the subModel to synch and have all needed data.

Reimplemented from Model.

References DataTransformModel::cache_submodel_responses(), DataTransformModel::collect_residuals(), ExperimentData::config_vars(), RecastModel::derived_synchronize(), DataTransformModel::expData, RecastModel::recastResponseMap, RecastModel::subModel, and Model::synchronize().

```cpp
const IntResponseMap & derived_synchronize_nowait() [protected], [virtual]
```

return any evaluations for which all experiment configurations have completed

Collect any completed subModel evals and build the residual sets for any fully completed evaluations. Like rekey functions in DakotaModel, but many sub-model to one recast-model. We do not force the subModel to synch.

Reimplemented from Model.

References DataTransformModel::cache_submodel_responses(), DataTransformModel::collect_residuals(), ExperimentData::config_vars(), RecastModel::derived_synchronize_nowait(), DataTransformModel::expData, RecastModel::recastResponseMap, RecastModel::subModel, and Model::synchronize().

```cpp
const IntResponseMap & filter_submodel_responses() [protected]
```

We don’t quite want the rekey behavior since multiple subModel evals map to one recast eval.)

References Model::cache_unmatched_response(), RecastModel::recastIdMap, RecastModel::subModel, and Model::synchronize().

Referenced by DataTransformModel::derived_evaluate().

```cpp
void transform_response_map(const IntResponseMap & submodel_resps, const Variables & recast_vars, Response & residual_resp) [protected]
```

transform a set of per-configuration subModel Responses to a single evaluation’s residuals

This transformation assumes the residuals are in submodel eval_id order.

References Dakota::abort_handler(), Model::current_variables(), DataTransformModel::expData, ExperimentData::form_residuals(), ExperimentData::num_experiments(), DataTransformModel::scale_response(), and RecastModel::subModel.

Referenced by DataTransformModel::collect_residuals(), and DataTransformModel::derived_evaluate().

```cpp
void set_mapping(const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set) [static], [protected]
```

map the inbound ActiveSet to the sub-model (map derivative variables)

**RecastModel** sets up a default set mapping before calling this update, so focus on updating the derivative variables vector.

References Model::cv(), ActiveSet::derivative_vector(), DataTransformModel::dtModelInstance, DataTransformModel::numHyperparams, ActiveSet::request_vector(), and RecastModel::subordinate_model().

Referenced by DataTransformModel::DataTransformModel().

```cpp
void init_continuous_vars() [protected]
```

Initialize continuous variable values/labels.

Pull up the continuous variable values and labels into the **RecastModel**, inserting the hyper-parameter values/labels.
References Model::all_continuous_variable(), Model::all_continuous_variable_label(), Model::all_continuous_variable_labels(), Model::all_continuous_variables(), SharedVariablesData::components_totals(), Model::current_variables(), DataTransformModel::expData, DataTransformModel::get_hyperparam_vc_index(), ExperimentData::hyperparam_labels(), DataTransformModel::numHyperparams, DataTransformModel::obsErrorMultiplierMode, Variables::shared_data(), RecastModel::subModel, and Dakota::svd().

Referenced by DataTransformModel::DataTransformModel().

```cpp
void expand_scales_array ( const SharedResponseData & srd, const T & submodel_array, size_t submodel_size, size_t recast_size, T & recast_array ) const [protected]
```

If size greater than 1, expand submodel_array by replicates to populate a pre-sized recast_array, otherwise copy.

Passing the inbound array size so we can use one function for Teuchos and std containers (size vs. length)

References DataTransformModel::expand_array().

Referenced by DataTransformModel::DataTransformModel().

13.37.4 Member Data Documentation

DataTransformModel * dtModelInstance [static], [protected]

static pointer to this class for use in static callbacks

initialization of static needed by RecastModel

Referenced by DataTransformModel::DataTransformModel(), DataTransformModel::primary_resp_difference(), and DataTransformModel::set_mapping().

The documentation for this class was generated from the following files:

- DataTransformModel.hpp
- DataTransformModel.cpp

13.38 DataVariables Class Reference

Handle class for variables specification data.

Public Member Functions

- **DataVariables ()**
  *constructor*

- **DataVariables (const DataVariables &)**
  *copy constructor*

- **~DataVariables ()**
  *destructor*

- **DataVariables operator= (const DataVariables &)**
  *assignment operator*

- **bool operator== (const DataVariables &)**
  *equality operator*

- **void write (std::ostream &s) const**
  *write a DataVariables object to an std::ostream*

- **void read (MPIUnpackBuffer &s)**
  *read a DataVariables object from a packed MPI buffer*
void write (MPIPackBuffer &s) const
    write a DataVariables object to a packed MPI buffer

DataVariablesRep * data_rep ()
    return dataVarsRep

size_t design ()
    return total number of design variables

size_t aleatory_uncertain ()
    return total number of aleatory uncertain variables

size_t epistemic_uncertain ()
    return total number of epistemic uncertain variables

size_t uncertain ()
    return total number of uncertain variables

size_t state ()
    return total number of state variables

size_t continuous_variables ()
    return total number of continuous variables

size_t discrete_variables ()
    return total number of discrete variables

size_t total_variables ()
    return total number of variables

Static Public Member Functions

- static bool id_compare (const DataVariables &dv, const std::string &id)
    compares the idVariables attribute of DataVariables objects

Private Attributes

- DataVariablesRep * dataVarsRep
    pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

13.38.1 Detailed Description

Handle class for variables specification data.

The DataVariables class is used to provide a memory management handle for the data in DataVariables-Rep. It is populated by IDRProblemDescDB::variables_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataVariables objects is maintained in ProblemDescDB::dataVariablesList, one for each variables specification in an input file.

The documentation for this class was generated from the following files:

- DataVariables.hpp
- DataVariables.cpp
13.39 DataVariablesRep Class Reference

Body class for variables specification data.

Public Attributes

- String idVariables
  
  string identifier for the variables specification data set (from the id_variables specification in VarSetId)

- short varsView

  user selection/override of variables view: {DEFAULT,ALL,DESIGN, UNCERTAIN,ALEATORY,UNCERTAIN,EPISTEMIC,UNCERTAIN,STATE}.VIEW

- short varsDomain

  user selection/override of variables domain: {DEFAULT,MIXED,RELAXED}.DOMAIN

- bool uncertainVarsInitPt

  flag indicating user specification of initial points (for local optimization-based UQ methods) for at least one uncertain variable type

- size_t numContinuousDesVars

  number of continuous design variables (from the continuous_design specification in VarDV)

- size_t numDiscreteDesRangeVars

  number of discrete design variables defined by an integer range (from the discrete_design_range specification in VarDV)

- size_t numDiscreteDesSetIntVars

  number of discrete design variables defined by a set of integers (from the discrete_design_set integer specification in VarDV)

- size_t numDiscreteDesSetStrVars

  number of discrete design variables defined by a set of strings (from the discrete_design_set string specification in VarDV)

- size_t numDiscreteDesSetRealVars

  number of discrete design variables defined by a set of reals (from the discrete_design_set real specification in VarDV)

- size_t numNormalUncVars

  number of normal uncertain variables (from the normal_uncertain specification in VarAUV)

- size_t numLognormalUncVars

  number of lognormal uncertain variables (from the lognormal_uncertain specification in VarAUV)

- size_t numUniformUncVars

  number of uniform uncertain variables (from the uniform_uncertain specification in VarAUV)

- size_t numLoguniformUncVars

  number of loguniform uncertain variables (from the loguniform_uncertain specification in VarAUV)

- size_t numTriangularUncVars

  number of triangular uncertain variables (from the triangular_uncertain specification in VarAUV)

- size_t numExponentialUncVars

  number of exponential uncertain variables (from the exponential_uncertain specification in VarAUV)

- size_t numBetaUncVars

  number of beta uncertain variables (from the beta_uncertain specification in VarAUV)

- size_t numGammaUncVars
number of gamma uncertain variables (from the `gamma_uncertain` specification in VarAUV)

- `size_t numGumbelUncVars`
  number of gumbel uncertain variables (from the `gumbel_uncertain` specification in VarAUV)

- `size_t numFrechetUncVars`
  number of frechet uncertain variables (from the `frechet_uncertain` specification in VarAUV)

- `size_t numWeibullUncVars`
  number of weibull uncertain variables (from the `weibull_uncertain` specification in VarAUV)

- `size_t numHistogramBinUncVars`
  number of histogram bin uncertain variables (from the `histogram_bin_uncertain` specification in VarAUV)

- `size_t numPoissonUncVars`
  number of Poisson uncertain variables (from the `poisson_uncertain` specification in VarAUV)

- `size_t numBinomialUncVars`
  number of binomial uncertain variables (from the `binomial_uncertain` specification in VarAUV)

- `size_t numNegBinomialUncVars`
  number of negative binomial uncertain variables (from the `negative_binomial_uncertain` specification in VarAUV)

- `size_t numGeometricUncVars`
  number of geometric uncertain variables (from the `geometric_uncertain` specification in VarAUV)

- `size_t numHyperGeomUncVars`
  number of hypergeometric uncertain variables (from the `hypergeometric_uncertain` specification in VarAUV)

- `size_t numHistogramPtIntUncVars`
  number of integer-valued histogram point uncertain variables (from the `histogram_point_uncertain` specification in VarAUV)

- `size_t numHistogramPtStrUncVars`
  number of string-valued histogram point uncertain variables (from the `histogram_point_uncertain` specification in VarAUV)

- `size_t numHistogramPtRealUncVars`
  number of real-valued histogram point uncertain variables (from the `histogram_point_uncertain` specification in VarAUV)

- `size_t numContinuousIntervalUncVars`
  number of continuous epistemic interval uncertain variables (from the `continuous_interval_uncertain` specification in VarEUV)

- `size_t numDiscreteIntervalUncVars`
  number of discrete epistemic interval uncertain variables (from the `discrete_interval_uncertain` specification in VarEUV)

- `size_t numDiscreteUncSetIntVars`
  number of discrete epistemic uncertain integer set variables (from the `discrete_uncertain_set_integer` specification in VarEUV)

- `size_t numDiscreteUncSetStrVars`
  number of discrete epistemic uncertain string set variables (from the `discrete_uncertain_set_string` specification in VarEUV)

- `size_t numDiscreteUncSetRealVars`
  number of discrete epistemic uncertain real set variables (from the `discrete_uncertain_set_real` specification in VarEUV)
• `size_t numContinuousStateVars`
  number of continuous state variables (from the `continuous_state` specification in `VarSV`)

• `size_t numDiscreteStateRangeVars`
  number of discrete state variables defined by an integer range (from the `discrete_state_range` specification in `VarDV`)

• `size_t numDiscreteStateSetIntVars`
  number of discrete state variables defined by a set of integers (from the `discrete_state_set` integer specification in `VarDV`)

• `size_t numDiscreteStateSetRealVars`
  number of discrete state variables defined by a set of reals (from the `discrete_state_set` real specification in `VarDV`)

• `RealVector continuousDesignVars`
  initial values for the continuous design variables array (from the `continuous_design_initial_point` specification in `VarDV`)

• `RealVector continuousDesignLowerBnds`
  lower bounds array for the continuous design variables (from the `continuous_design_lower_bounds` specification in `VarDV`)

• `RealVector continuousDesignUpperBnds`
  upper bounds array for the continuous design variables (from the `continuous_design_upper_bounds` specification in `VarDV`)

• `StringArray continuousDesignScaleTypes`
  scale types array for the continuous design variables (from the `continuous_design_scale_types` specification in `VarDV`)

• `RealVector continuousDesignScales`
  scales array for the continuous design variables (from the `continuous_design_scales` specification in `VarDV`)

• `IntVector discreteDesignRangeVars`
  initial values for the discrete design variables defined by an integer range (from the `discrete_design_range` initial_point specification in `VarDV`)

• `IntVector discreteDesignRangeLowerBnds`
  lower bounds array for the discrete design variables defined by an integer range (from the `discrete_design_range` lower_bounds specification in `VarDV`)

• `IntVector discreteDesignRangeUpperBnds`
  upper bounds array for the discrete design variables defined by an integer range (from the `discrete_design_range` upper_bounds specification in `VarDV`)

• `BitArray discreteDesignRangeCat`
  is each ddr var strictly categorical (true) or relaxable (false)

• `IntVector discreteDesignSetIntVars`
  initial values for the discrete design variables defined by an integer set (from the `discrete_design_set` integer initial_point specification in `VarDV`)

• `StringArray discreteDesignSetStrVars`
  initial values for the discrete design variables defined by a string set (from the `discrete_design_set` string initial_point specification in `VarDV`)
• RealVector discreteDesignSetRealVars
  initial values for the discrete design variables defined by a real set (from the discrete_design_set real initial_point specification in VarDV)

• IntSetArray discreteDesignSetInt
  complete set of admissible values for each of the discrete design variables defined by an integer set (from the discrete_design_set integer set_values specification in VarDV)

• StringSetArray discreteDesignSetStr
  complete set of admissible values for each of the discrete design variables defined by a string set (from the discrete_design_set string set_values specification in VarDV)

• RealSetArray discreteDesignSetReal
  complete set of admissible values for each of the discrete design variables defined by a real set (from the discrete-design_set real set_values specification in VarDV)

• BitArray discreteDesignSetIntCat
  is each ddsi var strictly categorical (true) or relaxable (false)

• BitArray discreteDesignSetRealCat
  is each ddsr var strictly categorical (true) or relaxable (false)

• RealMatrixArray discreteDesignSetIntAdj
  Adjacency matrices for each of the discrete design variables defined by an integer set (from the discrete_design_set integer categorical adjacency specification in VarDV)

• RealMatrixArray discreteDesignSetStrAdj
  Adjacency matrices for each of the discrete design variables defined by a string set (from the discrete_design_str set string adjacency specification in VarDV)

• RealMatrixArray discreteDesignSetRealAdj
  Adjacency matrices for each of the discrete design variables defined by a real set (from the discrete_design_set real categorical adjacency specification in VarDV)

• StringArray continuousDesignLabels
  labels array for the continuous design variables (from the continuous_design descriptors specification in VarDV)

• StringArray discreteDesignRangeLabels
  labels array for the discrete design variables defined by an integer range (from the discrete_design_range descriptors specification in VarDV)

• StringArray discreteDesignSetIntLabels
  labels array for the discrete design variables defined by an integer set (from the discrete_design_set int descriptors specification in VarDV)

• StringArray discreteDesignSetStrLabels
  labels array for the discrete design variables defined by a string set (from the discrete_design_set string descriptors specification in VarDV)

• StringArray discreteDesignSetRealLabels
  labels array for the discrete design variables defined by a real set (from the discrete_design_set real descriptors specification in VarDV)

• RealVector normalUncMeans
  means of the normal uncertain variables (from the means specification in VarCAUV_Normal)

• RealVector normalUncStdDevs
  standard deviations of the normal uncertain variables (from the std_deviations specification in VarCAUV_Normal)

• RealVector normalUncLowerBnds
CHAPTER 13. CLASS DOCUMENTATION

distribution lower bounds for the normal uncertain variables (from the lower_bounds specification in VarCAUV Normal)

- RealVector normalUncUpperBnds
distribution upper bounds for the normal uncertain variables (from the upper_bounds specification in VarCAUV Normal)

- RealVector normalUncVars
initial values of the normal uncertain variables (from the initial_point specification in VarCAUV Normal)

- RealVector lognormalUncLambdas
lambdas (means of the corresponding normals) of the lognormal uncertain variables (from the lambdas specification in VarCAUV_Lognormal)

- RealVector lognormalUncZetas
zetas (standard deviations of the corresponding normals) of the lognormal uncertain variables (from the zetas specification in VarCAUV_Lognormal)

- RealVector lognormalUncMeans
means of the lognormal uncertain variables (from the means specification in VarCAUV_Lognormal)

- RealVector lognormalUncStdDevs
standard deviations of the lognormal uncertain variables (from the std_deviations specification in VarCAUV_Lognormal)

- RealVector lognormalUncErrFacts
error factors for the lognormal uncertain variables (from the error_factors specification in VarCAUV_Lognormal)

- RealVector lognormalUncLowerBnds
distribution lower bounds for the lognormal uncertain variables (from the lower_bounds specification in VarCAUV_Lognormal)

- RealVector lognormalUncUpperBnds
distribution upper bounds for the lognormal uncertain variables (from the upper_bounds specification in VarCAUV_Lognormal)

- RealVector lognormalUncVars
initial values of the lognormal uncertain variables (from the initial_point specification in VarCAUV_Lognormal)

- RealVector uniformUncLowerBnds
distribution lower bounds for the uniform uncertain variables (from the lower_bounds specification in VarCAUV_Uniform)

- RealVector uniformUncUpperBnds
distribution upper bounds for the uniform uncertain variables (from the upper_bounds specification in VarCAUV_Uniform)

- RealVector uniformUncVars
initial values of the uniform uncertain variables (from the initial_point specification in VarCAUV_Uniform)

- RealVector loguniformUncLowerBnds
distribution lower bounds for the loguniform uncertain variables (from the lower_bounds specification in VarCAUV_Loguniform)

- RealVector loguniformUncUpperBnds
distribution upper bounds for the loguniform uncertain variables (from the upper_bounds specification in VarCAUV_Loguniform)

- RealVector loguniformUncVars
initial values of the loguniform uncertain variables (from the initial_point specification in VarCAUV_Loguniform)

- RealVector triangularUncModes
modes of the triangular uncertain variables (from the modes specification in VarCAUV_Triangular)

- RealVector triangularUncLowerBnds
  distribution lower bounds for the triangular uncertain variables (from the lower_bounds specification in VarCAUV_Triangular)

- RealVector triangularUncUpperBnds
  distribution upper bounds for the triangular uncertain variables (from the upper_bounds specification in VarCAUV_Triangular)

- RealVector triangularUncVars
  initial values of the triangular uncertain variables (from the initial_point specification in VarCAUV_Triangular)

- RealVector exponentialUncBetas
  beta factors for the exponential uncertain variables (from the betas specification in VarCAUV_Exponential)

- RealVector exponentialUncVars
  initial values of the exponential uncertain variables (from the initial_point specification in VarCAUV_Exponential)

- RealVector betaUncAlphas
  alpha factors for the beta uncertain variables (from the means specification in VarCAUV_Beta)

- RealVector betaUncBetas
  beta factors for the beta uncertain variables (from the std_deviations specification in VarCAUV_Beta)

- RealVector betaUncLowerBnds
  distribution lower bounds for the beta uncertain variables (from the lower_bounds specification in VarCAUV_Beta)

- RealVector betaUncUpperBnds
  distribution upper bounds for the beta uncertain variables (from the upper_bounds specification in VarCAUV_Beta)

- RealVector betaUncVars
  initial values of the beta uncertain variables (from the initial_point specification in VarCAUV_Beta)

- RealVector gammaUncAlphas
  alpha factors for the gamma uncertain variables (from the alphas specification in VarCAUV_Gamma)

- RealVector gammaUncBetas
  beta factors for the gamma uncertain variables (from the betas specification in VarCAUV_Gamma)

- RealVector gammaUncVars
  initial values of the gamma uncertain variables (from the initial_point specification in VarCAUV_Gamma)

- RealVector gumbelUncAlphas
  alpha factors for the gumbel uncertain variables (from the alphas specification in VarCAUV_Gumbel)

- RealVector gumbelUncBetas
  beta factors for the gumbel uncertain variables (from the betas specification in VarCAUV_Gumbel)

- RealVector gumbelUncVars
  initial values of the gumbel uncertain variables (from the initial_point specification in VarCAUV_Gumbel)

- RealVector frechetUncAlphas
  alpha factors for the frechet uncertain variables (from the alphas specification in VarCAUV_Frechet)

- RealVector frechetUncBetas
  beta factors for the frechet uncertain variables (from the betas specification in VarCAUV_Frechet)

- RealVector frechetUncVars
  initial values of the frechet uncertain variables (from the initial_point specification in VarCAUV_Frechet)
• **RealVector weibullUncAlphas**
  
  *alpha factors for the weibull uncertain variables (from the `alphas` specification in `VarCAUV_Weibull`)*

• **RealVector weibullUncBetas**

  *beta factors for the weibull uncertain variables (from the `betas` specification in `VarCAUV_Weibull`)*

• **RealVector weibullUncVars**

  *initial values of the weibull uncertain variables (from the `initial_point` specification in `VarCAUV_Weibull`)*

• **RealRealMapArray histogramUncBinPairs**

  *An array for each real-valued bin-based histogram uncertain variable. Each array entry is a map from a real value to its probability. (See continuous linear histogram in LHS manual; from the `histogram_bin_uncertain` specification in `VarCAUV_Bin_Histogram`). (x,y) ordinate specifications are converted to (x,c) counts within NIDR.*

• **RealVector histogramBinUncVars**

  *initial values of the histogram bin uncertain variables (from the `initial_point` specification in `VarCAUV_Bin_Histogram`)*

• **RealVector poissonUncLambdas**

  *lambdas (rate parameter) for the poisson uncertain variables (from the `lambdas` specification in `VarDAUV_Poisson`)*

• **IntVector poissonUncVars**

  *initial values of the poisson uncertain variables (from the `initial_point` specification in `VarDAUV_Poisson`)*

• **BitArray poissonUncCat**

  *is each poisson var strictly categorical (true) or relaxable (false)*

• **RealVector binomialUncProbPerTrial**

  *probabilities per each trial (p) for the binomial uncertain variables from the `prob_per_trial` specification in `VarDAUV_Binomial`)*

• **IntVector binomialUncNumTrials**

  *Number of trials (N) for the binomial uncertain variables from the `num_trials` specification in `VarDAUV_Binomial`)*

• **IntVector binomialUncVars**

  *initial values of the binomial uncertain variables (from the `initial_point` specification in `VarDAUV_Binomial`)*

• **BitArray binomialUncCat**

  *is each binomial var strictly categorical (true) or relaxable (false)*

• **RealVector negBinomialUncProbPerTrial**

  *probabilities per each trial (p) for the negative binomial uncertain variables from the `prob_per_trial` specification in `VarDAUV_Negative_Binomial`)*

• **IntVector negBinomialUncNumTrials**

  *Number of trials (N) for the negative binomial uncertain variables from the `num_trials` specification in `VarDAUV_Negative_Binomial`)*

• **IntVector negBinomialUncVars**

  *initial values of the negative binomial uncertain variables (from the `initial_point` specification in `VarDAUV_Negative_Binomial`)*

• **BitArray negBinomialUncCat**

  *is each negbinomial var strictly categorical (true) or relaxable (false)*

• **RealVector geometricUncProbPerTrial**

  *probabilities per each trial (p) for the geometric uncertain variables from the `prob_per_trial` specification in `VarDAUV_Geometric`)*

• **IntVector geometricUncVars**
initial values of the geometric uncertain variables (from the initial_point specification in VarDAUV_Geometric)

- BitArray geometricUncCat
  is each geometric var strictly categorical (true) or relaxable (false)

- IntVector hyperGeomUncTotalPop
  Size of total populations (N) for the hypergeometric uncertain variables from the total_population specification in VarDAUV_Hypergeometric

- IntVector hyperGeomUncSelectedPop
  Size of selected populations for the hypergeometric uncertain variables from the selected_population specification in VarDAUV_Hypergeometric

- IntVector hyperGeomUncNumDrawn
  Number failed in the selected populations for the hypergeometric variables from the num_drawn specification in VarDAUV_Hypergeometric

- IntVector hyperGeomUncVars
  initial values of the hypergeometric uncertain variables (from the initial_point specification in VarDAUV_Hypergeometric)

- BitArray hyperGeomUncCat
  is each hypergeom var strictly categorical (true) or relaxable (false)

- IntRealMapArray histogramUncPointIntPairs
  An array for each integer-valued point-based histogram uncertain variable. Each array entry is a map from an integer value to its probability. (See discrete histogram in LHS manual; from the histogram_point_uncertain specification in VarDAUV_Point_Histogram)

- IntVector histogramPointIntUncVars
  initial values of the real-valued histogram point uncertain variables (from the initial_point specification in VarDAUV_Point_Histogram)

- BitArray histogramUncPointIntCat
  is each hupi var strictly categorical (true) or relaxable (false)

- StringRealMapArray histogramUncPointStrPairs
  An array for each string-valued point-based histogram uncertain variable. Each array entry is a map from a string value to its probability. (See discrete histogram in LHS manual; from the histogram_point_uncertain specification in VarDAUV_Point_Histogram)

- StringArray histogramPointStrUncVars
  initial values of the real-valued histogram point uncertain variables (from the initial_point specification in VarDAUV_Point_Histogram)

- RealRealMapArray histogramUncPointRealPairs
  An array for each real-valued point-based histogram uncertain variable. Each array entry is a map from a real value to its probability. (See discrete histogram in LHS manual; from the histogram_point_uncertain specification in VarDAUV_Point_Histogram)

- RealVector histogramPointRealUncVars
  initial values of the real-valued histogram point uncertain variables (from the initial_point specification in VarDAUV_Point_Histogram)

- BitArray histogramUncPointRealCat
  is each hupr var strictly categorical (true) or relaxable (false)

- RealSymMatrix uncertainCorrelations
  correlation matrix for all uncertain variables (from the uncertain_correlation_matrix specification in VarDAUV_Correlations). This matrix specifies rank correlations for LHS sampling and correlation coefficients ($\rho_{ij} = \text{normalized covariance matrix}$) for other methods.
• RealRealPairRealMapArray \textit{continuousIntervalUncBasicProbs}
  
  Probability values per interval cell per epistemic interval uncertain variable (from the continuous_interval_uncertain_interval_probs specification in VarCEUV Interval)

• RealVector \textit{continuousIntervalUncVars}
  
  initial values of the continuous interval uncertain variables (from the initial_point specification in VarCEUV Interval)

• IntIntPairRealMapArray \textit{discreteIntervalUncBasicProbs}
  
  Probability values per interval cell per epistemic interval uncertain variable (from the discrete_interval_uncertain_interval_probs specification in VarDIUV)

• IntVector \textit{discreteIntervalUncVars}
  
  initial values of the discrete interval uncertain variables (from the initial_point specification in VarDIUV)

• BitArray \textit{discreteIntervalUncCat}
  
  is each diu var strictly categorical (true) or relaxable (false)

• IntRealMapArray \textit{discreteUncSetIntValuesProbs}
  
  complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by an integer set (from the discrete_uncertain_set_integer_set_values specification in VarDUSIV)

• IntVector \textit{discreteUncSetIntVars}
  
  initial values of the discrete uncertain set integer variables (from the initial_point specification in VarDUSIV)

• BitArray \textit{discreteUncSetIntCat}
  
  is each dusi var strictly categorical (true) or relaxable (false)

• StringRealMapArray \textit{discreteUncSetStrValuesProbs}
  
  complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by a string set (from the discrete_uncertain_set_string_set_values specification in VarDUSIV)

• StringArray \textit{discreteUncSetStrVars}
  
  initial values of the discrete uncertain set integer variables (from the initial_point specification in VarDUSIV)

• RealRealMapArray \textit{discreteUncSetRealValuesProbs}
  
  complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by a real set (from the discrete_uncertain_set_real_set_values specification in VarDUSRV)

• RealVector \textit{discreteUncSetRealVars}
  
  initial values of the discrete uncertain set real variables (from the initial_point specification in VarDUSRV)

• BitArray \textit{discreteUncSetRealCat}
  
  is each dusr var strictly categorical (true) or relaxable (false)

• RealVector \textit{continuousStateVars}
  
  initial values for the continuous state variables array (from the continuous_state_initial_point specification in VarSV)

• RealVector \textit{continuousStateLowerBnds}
  
  lower bounds array for the continuous state variables (from the continuous_state_lower_bounds specification in VarSV)

• RealVector \textit{continuousStateUpperBnds}
  
  upper bounds array for the continuous state variables (from the continuous_state_upper_bounds specification in VarSV)

• IntVector \textit{discreteStateRangeVars}
initial values for the discrete state variables defined by an integer range (from the discrete_state_range initial_point specification in VarSV)

- IntVector discreteStateRangeLowerBnds
  - lower bounds array for the discrete state variables defined by an integer range (from the discrete_state_range lower_bounds specification in VarSV)

- IntVector discreteStateRangeUpperBnds
  - upper bounds array for the discrete state variables defined by an integer range (from the discrete_state_range upper_bounds specification in VarSV)

- BitArray discreteStateRangeCat
  - is each dsr var strictly categorical (true) or relaxable (false)

- IntVector discreteStateSetIntVars
  - initial values for the discrete state variables defined by an integer set (from the discrete_state_set integer initial_point specification in VarSV)

- StringArray discreteStateSetStrVars
  - initial values for the discrete state variables defined by a string set (from the discrete_state_set string initial_point specification in VarSV)

- RealVector discreteStateSetRealVars
  - initial values for the discrete state variables defined by a real set (from the discrete_state_set real initial_point specification in VarSV)

- IntSetArray discreteStateSetInt
  - complete set of admissible values for each of the discrete state variables defined by an integer set (from the discrete_state_set integer set_values specification in VarSV)

- StringSetArray discreteStateSetStr
  - complete set of admissible values for each of the discrete state variables defined by a string set (from the discrete_state_set string set_values specification in VarSV)

- RealSetArray discreteStateSetReal
  - complete set of admissible values for each of the discrete state variables defined by a real set (from the discrete_state_set real set_values specification in VarSV)

- BitArray discreteStateSetIntCat
  - is each dssi var strictly categorical (true) or relaxable (false)

- BitArray discreteStateSetRealCat
  - is each dsr var strictly categorical (true) or relaxable (false)

- StringArray continuousStateLabels
  - labels array for the continuous state variables (from the continuous_state_descriptors specification in VarSV)

- StringArray discreteStateRangeLabels
  - labels array for the discrete state variables defined by an integer range (from the discrete_state_range descriptors specification in VarSV)

- StringArray discreteStateSetIntLabels
  - labels array for the discrete state variables defined by an integer set (from the discrete_state_set descriptors specification in VarSV)

- StringArray discreteStateSetStrLabels
  - labels array for the discrete state variables defined by a string set (from the discrete_state_set descriptors specification in VarSV)

- StringArray discreteStateSetRealLabels
labels array for the discrete state variables defined by a real set (from the `discrete_state_set descriptors` specification in `VarSV`)

- `IntVector discreteDesignSetIntLowerBnds`  
  discrete design integer set lower bounds inferred from set values
- `IntVector discreteDesignSetIntUpperBnds`  
  discrete design integer set upper bounds inferred from set values
- `StringArray discreteDesignSetStrLowerBnds`  
  discrete design string set lower bounds inferred from set values
- `StringArray discreteDesignSetStrUpperBnds`  
  discrete design string set upper bounds inferred from set values
- `RealVector discreteDesignSetRealLowerBnds`  
  discrete design real set lower bounds inferred from set values
- `RealVector discreteDesignSetRealUpperBnds`  
  discrete design real set upper bounds inferred from set values
- `RealVector continuousAleatoryUncVars`  
  array of values for all continuous aleatory uncertain variables
- `RealVector continuousAleatoryUncLowerBnds`  
  distribution lower bounds for all continuous aleatory uncertain variables (collected from `nuv_lower_bounds`, `lnuv_lower_bounds`, `uuv_lower_bounds`, `luuv_lower_bounds`, `tuv_lower_bounds`, and `buv_lower_bounds` specifications in `VarAUV`, and derived for gamma, gumbel, frechet, weibull and histogram bin specifications)
- `RealVector continuousAleatoryUncUpperBnds`  
  distribution upper bounds for all continuous aleatory uncertain variables (collected from `nuv_upper_bounds`, `lnuv_upper_bounds`, `uuv_upper_bounds`, `luuv_upper_bounds`, `tuv_lower_bounds`, and `buv_upper_bounds` specifications in `VarAUV`, and derived for gamma, gumbel, frechet, weibull and histogram bin specifications)
- `StringArray continuousAleatoryUncLabels`  
  labels for all continuous aleatory uncertain variables (collected from `nuv_descriptors`, `lnuv_descriptors`, `uuv_descriptors`, `luuv_descriptors`, `tuv_descriptors`, `buv_descriptors`, `gauv_descriptors`, `guuv_descriptors`, `fuv_descriptors`, `wuv_descriptors`, and `hbuv_descriptors` specifications in `VarAUV`)
- `IntVector discreteIntAleatoryUncVars`  
  array of values for all discrete integer aleatory uncertain variables
- `IntVector discreteIntAleatoryUncLowerBnds`  
  distribution lower bounds for all discrete integer aleatory uncertain variables
- `IntVector discreteIntAleatoryUncUpperBnds`  
  distribution upper bounds for all discrete integer aleatory uncertain variables
- `StringArray discreteIntAleatoryUncLabels`  
  labels for all discrete integer aleatory uncertain variables
- `StringArray discreteStrAleatoryUncVars`  
  array of values for all discrete string epistemic uncertain variables
- `StringArray discreteStrAleatoryUncLowerBnds`  
  distribution lower bounds for all discrete string epistemic uncertain variables
- `StringArray discreteStrAleatoryUncUpperBnds`  
  distribution upper bounds for all discrete string epistemic uncertain variables
13.39. DATAVARIABLESREP CLASS REFERENCE

- StringArray discreteStrAleatoryUncLabels
  labels for all discrete string aleatory uncertain variables
- RealVector discreteRealAleatoryUncVars
  array of values for all discrete real aleatory uncertain variables
- RealVector discreteRealAleatoryUncLowerBnds
  distribution lower bounds for all discrete real aleatory uncertain variables
- RealVector discreteRealAleatoryUncUpperBnds
  distribution upper bounds for all discrete real aleatory uncertain variables
- StringArray discreteRealAleatoryUncLabels
  labels for all discrete real aleatory uncertain variables
- RealVector continuousEpistemicUncVars
  array of values for all continuous epistemic uncertain variables
- RealVector continuousEpistemicUncLowerBnds
  distribution lower bounds for all continuous epistemic uncertain variables
- RealVector continuousEpistemicUncUpperBnds
  distribution upper bounds for all continuous epistemic uncertain variables
- StringArray continuousEpistemicUncLabels
  labels for all continuous epistemic uncertain variables
- IntVector discreteIntEpistemicUncVars
  array of values for all discrete integer epistemic uncertain variables
- IntVector discreteIntEpistemicUncLowerBnds
  distribution lower bounds for all discrete integer epistemic uncertain variables
- IntVector discreteIntEpistemicUncUpperBnds
  distribution upper bounds for all discrete integer epistemic uncertain variables
- StringArray discreteIntEpistemicUncLabels
  labels for all discrete integer epistemic uncertain variables
- StringArray discreteStrEpistemicUncVars
  array of values for all discrete string epistemic uncertain variables
- StringArray discreteStrEpistemicUncLowerBnds
  distribution lower bounds for all discrete string epistemic uncertain variables
- StringArray discreteStrEpistemicUncUpperBnds
  distribution upper bounds for all discrete string epistemic uncertain variables
- StringArray discreteStrEpistemicUncLabels
  labels for all discrete string epistemic uncertain variables
- RealVector discreteRealEpistemicUncVars
  array of values for all discrete real epistemic uncertain variables
- RealVector discreteRealEpistemicUncLowerBnds
  distribution lower bounds for all discrete real epistemic uncertain variables
- RealVector discreteRealEpistemicUncUpperBnds
  distribution upper bounds for all discrete real epistemic uncertain variables
- StringArray discreteRealEpistemicUncLabels
  labels for all discrete real epistemic uncertain variables
- IntVector discreteStateSetIntLowerBnds
CHAPTER 13. CLASS DOCUMENTATION

- IntVector discreteStateSetIntUpperBnds
- StringArray discreteStateSetStrLowerBnds
- StringArray discreteStateSetStrUpperBnds
- RealVector discreteStateSetRealLowerBnds
- RealVector discreteStateSetRealUpperBnds
- RealVector linearIneqConstraintCoeffs
- RealVector linearIneqLowerBnds
- RealVector linearIneqUpperBnds
- StringArray linearIneqScaleTypes
- RealVector linearIneqScales
- RealVector linearEqConstraintCoeffs
- RealVector linearEqTargets
- StringArray linearEqScaleTypes
- RealVector linearEqScales

Private Member Functions

- DataVariablesRep ()
  default constructor
- ~DataVariablesRep ()
  destructor
13.40. DDACEDesignCompExp Class Reference

Wrapper class for the DDACE design of experiments library.

Inheritance diagram for DDACEDesignCompExp:

```
  DDACEDesignCompExp
  |                     
  v                     
Analyzer
  |                     
  v                     
PStudyDACE
  |                     
  v                     
Iterator
```

- void write (std::ostream &s) const
  
  write a DataVariablesRep object to an std::ostream

- void read (MPIUnpackBuffer &s)
  
  read a DataVariablesRep object from a packed MPI buffer

- void write (MPIPackBuffer &s) const
  
  write a DataVariablesRep object to a packed MPI buffer

Private Attributes

- int referenceCount
  
  number of handle objects sharing dataVarsRep

Friends

- class DataVariables
  
  the handle class can access attributes of the body class directly

13.39.1 Detailed Description

Body class for variables specification data.

The DataVariablesRep class is used to contain the data from a variables keyword specification. Default values are managed in the DataVariablesRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataVariablesList is private.

The documentation for this class was generated from the following files:

- DataVariables.hpp
- DataVariables.cpp
Public Member Functions

- **DDACEDesignCompExp (ProblemDescDB &problem_db, Model &model)**
  primary constructor for building a standard DACE iterator
- **DDACEDesignCompExp (Model &model, int samples, int symbols, int seed, unsigned short sampling_method)**
  alternate constructor used for building approximations
- **~DDACEDesignCompExp ()**
  destructor
- **bool resize ()**
  reinitializes iterator based on new variable size

Protected Member Functions

- **void pre_run ()**
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- **void core_run ()**
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- **void post_input ()**
  read tabular data for post-run mode
- **void post_run (std::ostream &s)**
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- **int num_samples () const**
- **void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)**
  reset sampling iterator to use at least min_samples
- **unsigned short sampling_scheme () const**
  return sampling name
- **void vary_pattern (bool pattern_flag)**
  sets varyPattern in derived classes that support it
- **void get_parameter_sets (Model &model)**
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
- **void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix)**
  Generate one block of numSamples samples (ndim * num_samples), populating design_matrix.

Private Member Functions

- **void compute_main_effects ()**
  builds a DDaceMainEffects::OneWayANOVA if mainEffectsFlag is set
- **boost::shared_ptr < DDaceSamplerBase > create_sampler (Model &model)**
  create a DDACE sampler
- **void resolve_samples_symbols ()**
  convenience function for resolving number of samples and number of symbols from input.
### 13.40. DDACEDesignCompExp Class Reference

#### Private Attributes

- **unsigned short daceMethod**
  - oas, lhs, oa_lhs, random, box, jehnken, central_composite, or grid
- **int samplesSpec**
  - initial specification of number of samples
- **int symbolsSpec**
  - initial specification of number of symbols
- **int numSamples**
  - current number of samples to be evaluated
- **int numSymbols**
  - current number of symbols to be used in generating the sample set (inversely related to number of replications)
- **const int seedSpec**
  - the user seed specification for the random number generator (allows repeatable results)
- **int randomSeed**
  - current seed for the random number generator
- **bool allDataFlag**
  - flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_response()
- **size_t numDACERuns**
  - counter for number of executions for this object
- **bool varyPattern**
  - flag for continuing the random number sequence from a previous execution (e.g., for surrogate-based optimization) so that multiple executions are repeatable but not correlated.
- **bool mainEffectsFlag**
  - flag which specifies main effects
- **std::vector<std::vector<int>> symbolMapping**
  - mapping of symbols for main effects calculations

#### Additional Inherited Members

**13.40.1 Detailed Description**

Wrapper class for the DDACE design of experiments library.

The **DDACEDesignCompExp** class provides a wrapper for DDACE, a C++ design of experiments library from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. This class uses design and analysis of computer experiments (DACE) methods to sample the design space spanned by the bounds of a **Model**. It returns all generated samples and their corresponding responses as well as the best sample found.

**13.40.2 Constructor & Destructor Documentation**

**DDACEDesignCompExp ( ProblemDescDB & problem_db, Model & model )**

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

References Dakota::abort_handler(), DDACEDesignCompExp::daceMethod, DDACEDesignCompExp::main-EffectsFlag, Iterator::maxEvalConcurrency, Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer-::numDiscreteRealVars, Analyzer::numDiscreteStringVars, and DDACEDesignCompExp::numSamples.
DDACEDesignCompExp ( Model & model, int samples, int symbols, int seed, unsigned short sampling_method )

alternate constructor used for building approximations

This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.

References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, DDACEDesignCompExp::numSamples, and DDACEDesignCompExp::resolve_samples_symbols().

13.40.3 Member Function Documentation

void pre_run ( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Analyzer.

References DDACEDesignCompExp::get_parameter_sets(), Analyzer::get_vbd_parameter_sets(), Iterator::iteratedModel, DDACEDesignCompExp::numSamples, Analyzer::pre_run(), DDACEDesignCompExp::resolve_samples_symbols(), and PStudyDACE::varBasedDecompFlag.

void core_run ( ) [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References DDACEDesignCompExp::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, DDACEDesignCompExp::mainEffectsFlag, Analyzer::numLSqTerms, Analyzer::numObjFns, and Iterator::subIteratorFlag.

void post_run ( std::ostream & s ) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, SensAnalysisGlobal::compute_correlations(), DDACEDesignCompExp::compute_main_effects(), Analyzer::compute_vbd_stats(), DDACEDesignCompExp::create_sampler(), Iterator::iteratedModel, DDACEDesignCompExp::mainEffectsFlag, DDACEDesignCompExp::numSamples, Analyzer::post_run(), PStudyDACE::pStudyDACEsensGlobal, DDACEDesignCompExp::seedSpec, Iterator::subIteratorFlag, DDACEDesignCompExp::symbolMapping, and PStudyDACE::varBasedDecompFlag.
int num_samples() const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.
References DDACEDesignCompExp::numSamples.
Referenced by DDACEDesignCompExp::get_parameter_sets().

void resolve_samples_symbols() [private]

convenience function for resolving number of samples and number of symbols from input.

This function must define a combination of samples and symbols that is acceptable for a particular sampling algorithm. Users provide requests for these quantities, but this function must enforce any restrictions imposed by the sampling algorithms.

References Dakota::abort_handler(), DDACEDesignCompExp::daceMethod, Analyzer::numContinuousVars, DDACEDesignCompExp::numSamples, DDACEDesignCompExp::numSymbols, and Iterator::submethod_enum_to_string().

Referenced by DDACEDesignCompExp::DDACEDesignCompExp(), DDACEDesignCompExp::post_input(), and DDACEDesignCompExp::pre_run().

The documentation for this class was generated from the following files:
- DDACEDesignCompExp.hpp
- DDACEDesignCompExp.cpp

13.41 DirectApplicInterface Class Reference

Derived application interface class which spawns simulation codes and testers using direct procedure calls.

Inheritance diagram for DirectApplicInterface:

Public Member Functions

- DirectApplicInterface (const ProblemDescDB &problem_db)
  constructor
- ~DirectApplicInterface ()
  destructor
- void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.
- void derived_map_asynch (const ParamResponsePair &pair)
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.
- void wait_local_evaluations (PRPQueue &prp_queue)
For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- **void** test\_local\_evaluations (PRPQueue &prp\_queue)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- **int** synchronous\_local\_analysis (int analysis\_id)

- **const StringArray &** analysis\_drivers () const

  retrieve the analysis drivers specification for application interfaces

- **void** init\_communicators\_checks (int max\_eval\_concurrency)

- **void** set\_communicators\_checks (int max\_eval\_concurrency)

**Protected Member Functions**

- **virtual int** derived\_map\_if (const Dakota::String &if\_name)

  execute the input filter portion of a direct evaluation invocation

- **virtual int** derived\_map\_ac (const Dakota::String &ac\_name)

  execute an analysis code portion of a direct evaluation invocation

- **virtual int** derived\_map\_of (const Dakota::String &of\_name)

  execute the output filter portion of a direct evaluation invocation

- **virtual void** set\_local\_data (const Variables &vars, const ActiveSet &set)

  convenience function for local test simulators which sets per-evaluation variable and active set attributes; derived classes reimplementing this likely need to invoke the base class API

- **virtual void** set\_local\_data (const Response &response)

  convenience function for local test simulators which sets per-evaluation response attributes; derived classes reimplementing this likely need to invoke the base class API

- **virtual void** set\_local\_data (const Variables &vars, const ActiveSet &set, const Response &response)

  convenience function for local test simulators which sets per-evaluation variable, active set, and response attributes; derived classes reimplementing this likely need to invoke the base class API

- **void** overlay\_response (Response &response)

  convenience function for local test simulators which overlays response contributions from multiple analyses using MPI\_Reduce

**Protected Attributes**

- **String** iFilterName

  name of the direct function input filter

- **String** oFilterName

  name of the direct function output filter

- **driver\_t** iFilterType

  enum type of the direct function input filter

- **driver\_t** oFilterType

  enum type of the direct function output filter

- **bool** gradFlag

  signals use of fnGrads in direct simulator functions

- **bool** hessFlag
signals use of fnHessians in direct simulator functions

- `size_t numFns`
  number of functions in fnVals
- `size_t numVars`
  total number of continuous and discrete variables
- `size_t numACV`
  total number of continuous variables
- `size_t numADIV`
  total number of discrete integer variables
- `size_t numADRV`
  total number of discrete real variables
- `size_t numADSV`
  total number of discrete string variables
- `size_t numDerivVars`
  number of active derivative variables
- `unsigned short localDataView`
  bit-wise record of which local data views are active; see enum local_data_t
- `RealVector xC`
  continuous variables used within direct simulator fns
- `IntVector xDI`
  discrete int variables used within direct simulator fns
- `RealVector xDR`
  discrete real variables used within direct simulator fns
- `StringMultiArray xDS`
  discrete string variables used within direct simulator fns
- `StringMultiArray xCLabels`
  continuous variable labels
- `StringMultiArray xDILabels`
  discrete integer variable labels
- `StringMultiArray xDRLLabels`
  discrete real variable labels
- `StringMultiArray xDSLLabels`
  discrete string variable labels
- `std::map< var_t, varTypeMap > xCM`
  map from var_t enum to continuous value
- `std::map< var_t, int > xDIM`
  map from var_t enum to discrete int value
- `std::map< var_t, Real > xDRM`
  map from var_t enum to discrete real value
• std::map< var_t, String > xDSM
  
  map from var_t enum to discrete string value

• std::vector< var_t > varTypeDVV
  var_t enumerations corresponding to DVV components

• std::vector< var_t > xCMLLabels
  var_t enumerations corresponding to continuous variable labels

• std::vector< var_t > xDIMLabels
  var_t enumerations corresponding to discrete integer variable labels

• std::vector< var_t > xDRMLabels
  var_t enumerations corresponding to discrete real variable labels

• std::vector< var_t > xDSMLabels
  var_t enumerations corresponding to discrete string variable labels

• ShortArray directFnASV
  class scope active set vector

• SizetArray directFnDVV
  class scope derivative variables vector

• RealVector fnVals
  response fn values within direct simulator fns

• RealMatrix fnGrads
  response fn gradients w/i direct simulator fns

• RealSymMatrixArray fnHessians
  response fn Hessians within direct fns

• StringArray analysisDrivers
  the set of analyses within each function evaluation (from the analysis_drivers interface specification)

• std::vector< driver_t > analysisDriverTypes
  conversion of analysisDrivers to driver_t

• size_t analysisDriverIndex
  the index of the active analysis driver within analysisDrivers

• String2DArray analysisComponents
  the set of optional analysis components used by the analysis drivers (from the analysis_components interface specification)

Private Member Functions

• void map_labels_to_enum (StringMultiArrayConstView &src, std::vector< var_t > &dest)
  map labels in src to var_t in dest

13.41.1 Detailed Description

Derived application interface class which spawns simulation codes and testers using direct procedure calls.

DirectApplicInterface uses a few linkable simulation codes and several internal member functions to perform parameter to response mappings.
13.42. DISCREPANCYCORRECTION CLASS REFERENCE

13.41.2 Member Function Documentation

int synchronous_local_analysis ( int analysis_id ) [inline], [virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().
  Reimplemented from ApplicationInterface.
  References DirectApplicInterface::analysisDriverIndex, DirectApplicInterface::analysisDrivers, and DirectApplicInterface::derived_map_ac().

void init_communicators_checks ( int max_eval_concurrency ) [inline], [virtual]

Process init issues as warnings since some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used and DirectApplicInterface allows override by derived plug-ins.
  Reimplemented from ApplicationInterface.
  References ApplicationInterface::check_asynchronous(), and ApplicationInterface::check_multiprocessor_asynchronous().

void set_communicators_checks ( int max_eval_concurrency ) [inline], [virtual]

Process run-time issues as hard errors.
  Reimplemented from ApplicationInterface.
  Reimplemented in SerialDirectApplicInterface, and ParallelDirectApplicInterface.
  References Dakota::abort_handler(), ApplicationInterface::check_asynchronous(), and ApplicationInterface::check_multiprocessor_asynchronous().

int derived_map_ac ( const Dakota::String & ac_name ) [protected], [virtual]

execute an analysis code portion of a direct evaluation invocation
  When a direct analysis/filter is a member function, the (vars,set,response) data does not need to be passed through the API. If, however, non-member analysis/filter functions are added, then pass (vars,set,response) through to the non-member fns:

// API declaration
int sim(const Variables& vars, const ActiveSet& set, Response& response);
// use of API within derived_map_ac()
if (ac_name == "sim")
  fail_code = sim(directFnVars, directFnActSet, directFnResponse);

  Reimplemented in SerialDirectApplicInterface, ParallelDirectApplicInterface, MatlabInterface, PythonInterface, TestDriverInterface, and ScilabInterface.
  References Dakota::abort_handler(), and ApplicationInterface::analysisServerId.
  Referenced by DirectApplicInterface::derived_map(), and DirectApplicInterface::synchronous_local_analysis().
  The documentation for this class was generated from the following files:

- DirectApplicInterface.hpp
- DirectApplicInterface.cpp

13.42 DiscrepancyCorrection Class Reference

Base class for discrepancy corrections.
Public Member Functions

- **DiscrepancyCorrection ()**
  - default constructor
- **DiscrepancyCorrection (Model &surr_model, const IntSet &surr_fn_indices, short corr_type, short corr_order)**
  - standard constructor
- **DiscrepancyCorrection (const IntSet &surr_fn_indices, size_t num_fns, size_t num_vars, short corr_type, short corr_order)**
  - alternate constructor
- **~DiscrepancyCorrection ()**
  - destructor
- **void initialize (Model &surr_model, const IntSet &surr_fn_indices, short corr_type, short corr_order)**
  - initialize the DiscrepancyCorrection data
- **void initialize (const IntSet &surr_fn_indices, size_t num_fns, size_t num_vars, short corr_type, short corr_order)**
  - initialize the DiscrepancyCorrection data
- **void initialize (const IntSet &surr_fn_indices, size_t num_fns, size_t num_vars, short corr_type, short corr_order, const String &approx_type)**
  - initialize the DiscrepancyCorrection data
- **void compute (const Variables &vars, const Response &truth_response, const Response &approx_response, bool quiet_flag=false)**
  - compute the correction required to bring approx_response into agreement with truth_response and store in \{add,mult\} Corrections
- **void compute (const Response &truth_response, const Response &approx_response, Response &discrepancy_response, bool quiet_flag=false)**
  - compute the correction required to bring approx_response into agreement with truth_response and store in discrepancy_response
- **void compute (const VariablesArray &vars_array, const ResponseArray &truth_response_array, const ResponseArray &approx_response, bool quiet_flag=false)**
  - compute the correction required to bring approx_response into agreement with truth_response as a function of the variables and store in \{add,mult\} Corrections
- **void apply (const Variables &vars, Response &approx_response, bool quiet_flag=false)**
  - apply the correction computed in compute() to approx_response
- **void compute_variance (const VariablesArray &vars_array, RealMatrix &approx_variance, bool quiet_flag=false)**
  - compute the variance of approx_response
- **short correction_type () const**
  - return correctionType
- **short correction_order () const**
  - return correctionOrder
- **short data_order () const**
  - return dataOrder
- **bool computed () const**
  - return correctionComputed
- **bool initialized () const**
  - return initializedFlag
13.42. DISCREPANCYCORRECTION CLASS REFERENCE

Protected Attributes

- IntSet surrogateFnIndices
  for mixed response sets, this array specifies the response function subset that is approximated
- bool initializedFlag
  indicates that discrepancy correction instance has been initialized following construction
- short correctionType
  approximation correction approach to be used: NO_CORRECTION, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, or COMBINED_CORRECTION.
- short correctionOrder
  approximation correction order to be used: 0, 1, or 2
- short dataOrder
  order of correction data in 3-bit format: overlay of 1 (value), 2 (gradient), and 4 (Hessian)
- bool correctionComputed
  flag indicating whether or not a correction has been computed and is available for application
- size_t numFns
  total number of response functions (of which surrogateFnIndices may define a subset)
- size_t numVars
  number of continuous variables active in the correction

Private Member Functions

- void initialize_corrections ()
  internal convenience function shared by overloaded initialize() variants
- bool check_scaling (const RealVector &truth_fns, const RealVector &approx_fns)
  define badScalingFlag
- void compute_additive (const Response &truth_response, const Response &approx_response, int index, Real &discrep_fn, RealVector &discrep_grad, RealSymMatrix &discrep_hess)
  internal convenience function for computing additive corrections between truth and approximate responses
- void compute_multiplicative (const Response &truth_response, const Response &approx_response, int index, Real &discrep_fn, RealVector &discrep_grad, RealSymMatrix &discrep_hess)
  internal convenience function for computing multiplicative corrections between truth and approximate responses
- void apply_additive (const Variables &vars, Response &approx_response)
  internal convenience function for applying additive corrections to an approximate response
- void apply_multiplicative (const Variables &vars, Response &approx_response)
  internal convenience function for applying multiplicative corrections to an approximate response
- void apply_additive (const Variables &vars, RealVector &approx_fns)
  internal convenience function for applying additive corrections to a set of response functions
- void apply_multiplicative (const Variables &vars, RealVector &approx_fns)
  internal convenience function for applying multiplicative corrections to a set of response functions
- const Response & search_db (const Variables &search_vars, const ShortArray &search_asv)
  search data pairs for missing approximation data
Private Attributes

- **bool badScalingFlag**
  flag used to indicate function values near zero for multiplicative corrections; triggers an automatic switch to additive corrections

- **bool computeAdditive**
  flag indicating the need for additive correction calculations

- **bool computeMultiplicative**
  flag indicating the need for multiplicative correction calculations

- **String approxType**
  string indicating the discrepancy approximation type

- **SharedApproxData sharedData**
  data that is shared among all correction Approximations

- **std::vector< Approximation > addCorrections**
  array of additive corrections; surrogate models of a model discrepancy function (formed from model differences)

- **std::vector< Approximation > multCorrections**
  array of multiplicative corrections; surrogate models of a model discrepancy function (formed from model ratios)

- **Model surrModel**
  shallow copy of the surrogate model instance as returned by Model::surrogate_model() (the DataFitSurrModel or HierarchSurrModel::lowFidelityModel instance)

- **RealVector combineFactors**
  factors for combining additive and multiplicative corrections. Each factor is the weighting applied to the additive correction and 1.-factor is the weighting applied to the multiplicative correction. The factor value is determined by an additional requirement to match the high fidelity function value at the previous correction point (e.g., previous trust region center). This results in a multipoint correction instead of a strictly local correction.

- **Variables correctionPrevCenterPt**
  copy of center point from the previous correction cycle

- **RealVector truthFnsCenter**
  truth function values at the current correction point

- **RealVector approxFnsCenter**
  Surrogate function values at the current correction point.

- **RealMatrix approxGradsCenter**
  Surrogate gradient values at the current correction point.

- **RealVector truthFnsPrevCenter**
  copy of truth function values at center of previous correction cycle

- **RealVector approxFnsPrevCenter**
  copy of approximate function values at center of previous correction cycle

13.42.1 Detailed Description

Base class for discrepancy corrections.

The DiscrepancyCorrection class provides common functions for computing and applying corrections to approximations.
### 13.42.2 Member Function Documentation

```cpp
void compute ( const Variables & vars, const Response & truth_response, const Response & approx_response, bool quiet_flag = false )
```

compute the correction required to bring approx_response into agreement with truth_response and store in \{add,mult\} Corrections

Compute an additive or multiplicative correction that corrects the approx_response to have 0th-order consistency (matches values), 1st-order consistency (matches values and gradients), or 2nd-order consistency (matches values, gradients, and Hessians) with the truth_response at a single point (e.g., the center of a trust region). The 0th-order, 1st-order, and 2nd-order corrections use scalar values, linear scaling functions, and quadratic scaling functions, respectively, for each response function.

References Response::active_set(), DiscrepancyCorrection::addCorrections, DiscrepancyCorrection::apply(), DiscrepancyCorrection::apply_additive(), DiscrepancyCorrection::apply_multiplicative(), DiscrepancyCorrection::approxFnsCenter, DiscrepancyCorrection::approxFnsPrevCenter, DiscrepancyCorrection::approxGradsCenter, DiscrepancyCorrection::approxType, DiscrepancyCorrection::badScalingFlag, DiscrepancyCorrection::check_scaling(), DiscrepancyCorrection::combineFactors, DiscrepancyCorrection::compute_additive(), DiscrepancyCorrection::compute_multiplicative(), DiscrepancyCorrection::computeAdditive, DiscrepancyCorrection::computeMultiplicative, Variables::continuous_variables(), Response::copy(), DiscrepancyCorrection::correctionComputed, DiscrepancyCorrection::correctionOrder, DiscrepancyCorrection::correctionPrevCenterPt, DiscrepancyCorrection::correctionType, DiscrepancyCorrection::dataOrder, Variables::discrete_int_variables(), Variables::discrete_real_variables(), Response::function_gradients(), Response::function_values(), Model::is_null(), DiscrepancyCorrection::multCorrections, DiscrepancyCorrection::numFns, DiscrepancyCorrection::numVars, ActiveSet::request_values(), DiscrepancyCorrection::sharedData, DiscrepancyCorrection::surrModel, DiscrepancyCorrection::surrogateFnIndices, DiscrepancyCorrection::truthFnsCenter, and DiscrepancyCorrection::truthFnsPrevCenter.

Referenced by NonDBayesCalibration::build_model_discrepancy(), DiscrepancyCorrection::compute(), DataFitSurrModel::derived_evaluate(), DataFitSurrModel::derived_synchronize(), and DataFitSurrModel::derived_synchronize_nowait().

The documentation for this class was generated from the following files:

- DiscrepancyCorrection.hpp
- DiscrepancyCorrection.cpp

### 13.43 DLSSolverTraits Class Reference

A version of TraitsBase specialized for DLSSolver.

Inheritance diagram for DLSSolverTraits:

```
traitsBase
|----------------|
| DLSSolverTraits
```

#### Public Member Functions

- `DLSSolverTraits ()`
  
  *default constructor*

- `virtual ~DLSSolverTraits ()`
  
  *destructor*
• virtual bool is\_derived ()
  
  A temporary query used in the refactor.
• bool supports\_continuous\_variables ()
  
  Return the flag indicating whether method supports continuous variables.
• bool supports\_linear\_equality ()
  
  Return the flag indicating whether method supports linear equalities.
• bool supports\_linear\_inequality ()
  
  Return the flag indicating whether method supports linear inequalities.
• bool supports\_nonlinear\_equality ()
  
  Return the flag indicating whether method supports nonlinear equalities.
• bool supports\_nonlinear\_inequality ()
  
  Return the flag indicating whether method supports nonlinear inequalities.
• NONLINEAR\_INEQUALITY\_FORMAT nonlinear\_inequality\_format ()
  
  Return the format used for nonlinear inequality constraints.

13.43.1 Detailed Description

A version of TraitsBase specialized for DLSolver.

The documentation for this class was generated from the following file:

• DLSolver.hpp

13.44 DOTTraits Class Reference

Wrapper class for the DOT optimization library.

Inheritance diagram for DOTTraits:

```
TraitsBase
   ^
   |  
DOTTraits
```

Public Member Functions

• DOTTraits ()
  
  default constructor
• virtual ~DOTTraits ()
  
  destructor
• virtual bool is\_derived ()
  
  A temporary query used in the refactor.
• bool supports\_continuous\_variables ()
  
  Return the flag indicating whether method supports continuous variables.
• bool supports\_linear\_equality ()
  
  Return the flag indicating whether method supports linear equalities.
13.45. **JEGAOPTIMIZER::DRIVER CLASS REFERENCE**

- `bool supports_linear_inequality()`  
  *Return the flag indicating whether method supports linear inequalities.*
- `LINEAR_INEQUALITY_FORMAT linear_inequality_format()`  
  *Return the format used for linear inequality constraints.*
- `bool supports_nonlinear_inequality()`  
  *Return the flag indicating whether method supports nonlinear equalities.*
- `NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format()`  
  *Return the format used for nonlinear inequality constraints.*

### 13.44.1 Detailed Description

Wrapper class for the DOT optimization library.

The DOTOptimizer class provides a wrapper for DOT, a commercial Fortran 77 optimization library from Vanderplaats Research and Development. It uses a reverse communication mode, which avoids the static member function issues that arise with function pointer designs (see NPSOLOptimizer and SNLLOptimizer).

The user input mappings are as follows: `max_iterations` is mapped into DOT’s ITMAX parameter within its IPRM array, `max_function_evaluations` is implemented directly in the core run() loop since there is no DOT parameter equivalent, `convergence_tolerance` is mapped into DOT’s DELOBJ parameter (the relative convergence tolerance) within its RPRM array, `output_verbosity` is mapped into DOT’s IPRINT parameter within its function call parameter list (verbose: IPRINT = 7; quiet: IPRINT = 3), and `optimization_type` is mapped into DOT’s MINMAX parameter within its function call parameter list. Refer to [Vanderplaats Research and Development, 1995] for information on IPRM, RPRM, and the DOT function call parameter list. A version of TraitsBase specialized for DOT optimizers.

The documentation for this class was generated from the following file:

- DOTOptimizer.hpp

### 13.45 JEGAOptimizer::Driver Class Reference

A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm. Inherits Driver.

#### Public Member Functions

- `GeneticAlgorithm * ExtractAllData (const AlgorithmConfig &algConfig)`  
  *Reads all required data from the problem description database stored in the supplied algorithm config.*
- `DesignOFSortSet PerformIterations (GeneticAlgorithm *theGA)`  
  *Performs the required iterations on the supplied GA.*
- `void DestroyAlgorithm (GeneticAlgorithm *theGA)`  
  *Deletes the supplied GA.*
- `Driver (const ProblemConfig &probConfig)`  
  *Default constructs a Driver.*
13.45.1 Detailed Description

A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm. This is necessary because DAKOTA requires that all problem information be extracted from the problem description DB at the time of Optimizer construction and the front end does it all in the execute algorithm method which must be called in core_run.

13.45.2 Constructor & Destructor Documentation

Driver ( const ProblemConfig & probConfig ) [inline]

Default constructs a Driver.

Parameters

| probConfig | The definition of the problem to be solved by this Driver whenever ExecuteAlgorithm is called. |

The problem can be solved in multiple ways by multiple algorithms even using multiple different evaluators by issuing multiple calls to ExecuteAlgorithm with different AlgorithmConfigs.

13.45.3 Member Function Documentation

GeneticAlgorithm* ExtractAllData ( const AlgorithmConfig & algConfig ) [inline]

Reads all required data from the problem description database stored in the supplied algorithm config.

The returned GA is fully configured and ready to be run. It must also be destroyed at some later time. You MUST call DestroyAlgorithm for this purpose. Failure to do so could result in a memory leak and an eventual segmentation fault! Be sure to call DestroyAlgorithm prior to destroying the algorithm config that was used to create it!

This is just here to expose the base class method to users.

Parameters

| algConfig | The fully loaded configuration object containing the database of parameters for the algorithm to be run on the known problem. |

Returns

The fully configured and loaded GA ready to be run using the PerformIterations method.

Referenced by JEGAOptimizer::core_run().

DesignOFSortSet PerformIterations ( GeneticAlgorithm* theGA ) [inline]

Performs the required iterations on the supplied GA.

This includes the calls to AlgorithmInitialize and AlgorithmFinalize and logs some information if appropriate.

This is just here to expose the base class method to users.

Parameters

| theGA | The GA on which to perform iterations. This parameter must be non-null. |

Returns

The final solutions reported by the supplied GA after all iterations and call to AlgorithmFinalize.

Referenced by JEGAOptimizer::core_run().
void DestroyAlgorithm ( GeneticAlgorithm ∗ theGA ) [inline]

Deletes the supplied GA.

Use this method to destroy a GA after all iterations have been run. This method knows if the log associated with the GA was created here and needs to be destroyed as well or not.

This is just here to expose the base class method to users.

Be sure to use this prior to destroying the algorithm config object which contains the target. The GA destructor needs the target to be in tact.

Parameters

| theGA | The algorithm that is no longer needed and thus must be destroyed. |

Referenced by JEGAOptimizer::core_run().

The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp

13.46  EffGlobalMinimizer Class Reference

Inheritance diagram for EffGlobalMinimizer:

```
             Iterator
               ↓
              Minimizer
                 ↓
             SurrBasedMinimizer
                ↓
       EffGlobalMinimizer
```

Public Member Functions

- EffGlobalMinimizer (ProblemDescDB &problem_db, Model &model)
  standard constructor
- ~EffGlobalMinimizer ()
  alternate constructor for instantiations "on the fly"
- void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- const Model & algorithm_space_model () const

Private Member Functions

- void minimize_surrogates_on_model ()
  called by minimize_surrogates for setUpType == "model"
- void get_best_sample ()
  called by minimize_surrogates for setUpType == "user_functions"
- Real expected_improvement (const RealVector &means, const RealVector &variances)
expected improvement function for the GP

- RealVector expected_violation (const RealVector &means, const RealVector &variances)

  expected violation function for the constraint functions

- void update_penalty ()

  initialize and update the penaltyParameter

Static Private Member Functions

- static void EIF_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)

  static function used as the objective function in the Expected Improvement (EIF) problem formulation for PMA

Private Attributes

- String setUpType

  controls iteration mode: "model" (normal usage) or "user functions" (user-supplied functions mode for "on the fly" instantiations).

- Model fHatModel

  GP model of response, one approximation per response function.

- Model eifModel

  recast model which assimilates mean and variance to solve the max(EIF) sub-problem

- Real meritFnStar

  minimum penalized response from among true function evaluations

- RealVector truthFnStar

  true function values corresponding to the minimum penalized response

- RealVector varStar

  point that corresponds to the optimal value meritFnStar

- short dataOrder

  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec

Static Private Attributes

- static EffGlobalMinimizer * effGlobalInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

13.46.1 Detailed Description

The EffGlobalMinimizer class provides an implementation of the Efficient Global Optimization algorithm developed by Jones, Schonlau, & Welch as well as adaptation of the concept to nonlinear least squares.
13.46.2 Constructor & Destructor Documentation

```cpp
~EffGlobalMinimizer()
```
alternate constructor for instantiations "on the fly"

destructor

13.46.3 Member Function Documentation

```cpp
void core_run() [virtual]
```
core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Dakota::abort_handler(), EffGlobalMinimizer::minimize_surrogates_on_model(), and EffGlobalMinimizer::setUpType.

```cpp
const Model & algorithm_space_model() const [inline], [virtual]
```
default definition that gets redefined in selected derived Minimizers

Reimplemented from Minimizer.

References EffGlobalMinimizer::fHatModel.

```cpp
void get_best_sample() [private]
```
called by minimize_surrogates for setUpType == "user_functions"

determine best solution from among sample data for expected improvement function

References Model::approximation_data(), SurrBasedMinimizer::augmented_lagrangian_merit(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), Model::evaluate(), EffGlobalMinimizer::fHatModel, Response::function_values(), Iterator::iteratedModel, EffGlobalMinimizer::meritFnStar, Minimizer::numFunctions, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), EffGlobalMinimizer::truthFnStar, and EffGlobalMinimizer::varStar.

Referenced by EffGlobalMinimizer::minimize_surrogates_on_model().

The documentation for this class was generated from the following files:

- EffGlobalMinimizer.hpp
- EffGlobalMinimizer.cpp

13.47 EffGlobalTraits Class Reference

Implementation of Efficient Global Optimization/Least Squares algorithms.

Inheritance diagram for EffGlobalTraits:

```
  TraitsBase
     |     |
     |     +-- EffGlobalTraits
```

Public Member Functions

- **EffGlobalTraits ()**
  
  *default constructor*

- **virtual ~EffGlobalTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

13.47.1 Detailed Description

Implementation of Efficient Global Optimization/Least Squares algorithms. A version of TraitsBase specialized for efficient global minimizer

The documentation for this class was generated from the following file:

- **EffGlobalMinimizer.hpp**

13.48 EmbedHybridMetaIterator Class Reference

Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.

Inheritance diagram for EmbedHybridMetaIterator:

```
        Iterator
         |
         V
      MetaIterator
         |
         V
EmbedHybridMetaIterator
```

Public Member Functions

- **EmbedHybridMetaIterator (ProblemDescDB &problem_db)**
  
  *standard constructor*

- **EmbedHybridMetaIterator (ProblemDescDB &problem_db, Model &model)**
  
  *alternate constructor*

- **~EmbedHybridMetaIterator ()**
  
  *destructor*
Protected Member Functions

- void core_run ()
  Performs the hybrid iteration by executing global and local iterators, using a set of models that may vary in fidelity.
- void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- IntIntPair estimate_partition_bounds ()
  estimate the minimum and maximum partition sizes that can be utilized by this Iterator
- const Variables & variables_results () const
  return the final solution from the embedded hybrid (variables)
- const Response & response_results () const
  return the final solution from the embedded hybrid (response)

Private Attributes

- Iterator globalIterator
  the top-level outer iterator (e.g., global minimizer)
- Model globalModel
  the model employed by the top-level outer iterator
- Iterator localIterator
  the inner iterator (e.g., local minimizer)
- Model localModel
  the model employed by the inner iterator
- bool singlePassedModel
  use of constructor that enforces use of a single passed Model
- Real localSearchProb
  the probability of running a local search refinement within phases of the global minimization for tightly-coupled hybrids

Additional Inherited Members

13.48.1 Detailed Description

Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.

This meta-iterator uses multiple methods in close coordination, generally using a local search minimizer repeatedly within a global minimizer (the local search minimizer refines candidate minima which are fed back to the global minimizer).

The documentation for this class was generated from the following files:

- EmbedHybridMetaIterator.hpp
- EmbedHybridMetaIterator.cpp
13.49 Environment Class Reference

Base class for the environment class hierarchy.

Inheritance diagram for Environment:

```
Environment
  ↓
ExecutableEnvironment  LibraryEnvironment
```

Public Member Functions

- **Environment ()**
  - default constructor: empty envelope
- **Environment (int argc, char ∗argv[])**
  - envelope constructor for ExecutableEnvironment letter
- **Environment (ProgramOptions prog_opts)**
- **Environment (MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts=ProgramOptions())**
- **Environment (const String &env_type)**
  - envelope constructor for letter type identified by String
- **Environment (const Environment &env)**
  - copy constructor
- **virtual ~Environment ()**
  - destructor
- **Environment operator= (const Environment &env)**
  - assignment operator
- **virtual void execute ()**
  - the run function for the environment: invoke the iterator(s) on the model(s). Called from main.cpp.
- **bool check () const**
  - Print status of check and return true if in a "check" mode, including version and help. Return false if proceeding to a run mode.
- **MPIManager & mpi_manager ()**
  - return mpiManager
- **ProgramOptions & program_options ()**
  - return programOptions
- **OutputManager & output_manager ()**
  - return outputManager
- **ParallelLibrary & parallel_library ()**
  - return parallelLib
- **ProblemDescDB & problem_description_db ()**
  - return probDescDB
- **const Variables & variables_results () const**
  - return the final environment solution (variables)
const Response & response_results () const
return the final environment solution (response)

void exit_mode (const String &mode="exit")
allow environment clients to set Dakota exit behavior (throw vs. exit)

Protected Member Functions

- Environment (BaseConstructor)
  constructor initializes the base class part of default-constructed letters
- Environment (BaseConstructor, int argc, char ∗argv[ ])
  constructor initializes the base class part of executable letter classes
- Environment (BaseConstructor, ProgramOptions prog_opts, MPI_Comm dakota_mpi_comm=MPI_COMM_WORLD)
  constructor initializes the base class part of library letter classes
- void parse (bool check_bcast_database=true, DbCallbackFunctionPtr callback=NULL, void ∗callback_data=NULL)
  parse inputs, callbacks, and optionally check and broadcast
- void construct ()
  Instantiate topLevelIterator.
- void destruct ()
  Deallocate parallel partitioning for topLevelIterator.
- bool assign_model_pointer () const

Protected Attributes

- MPIManager mpiManager
  the MPI manager instance
- ProgramOptions programOptions
  the command line options manager
- OutputManager outputManager
  (tagged) output stream manager
- ParallelLibrary parallelLib
  the parallel library instance
- ProblemDescDB probDescDB
  the parser database instance
- Iterator topLevelIterator
  the top level (meta-)iterator
- UsageTracker usageTracker
  tool for Dakota usage tracking (this is a thin wrapper class)

Private Member Functions

- Environment * get_environment (const String &env_type)
  Used by the envelope to instantiate the correct letter class.
Private Attributes

- Environment * environmentRep
  pointer to the letter (initialized only for the envelope)
- int referenceCount
  number of objects sharing environmentRep

13.49.1 Detailed Description

Base class for the environment class hierarchy.

The Environment class is the base class for the class hierarchy providing the top level control in DAKOTA. The environment is responsible for creating and managing iterators and models. For memory efficiency and enhanced polymorphism, the environment hierarchy employs the "letter/envelope idiom" (see Coplien “Advanced C++”, p. 133), for which the base class (Environment) serves as the envelope and one of the derived classes (selected in Environment::get_environment()) serves as the letter.

13.49.2 Constructor & Destructor Documentation

Environment ( )
default constructor: empty envelope
  Default envelope constructor. environmentRep is NULL in this case, which makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

Environment ( int argc, char * argv[ ] )
envelope constructor for ExecutableEnvironment letter
  Envelope constructor for ExecutableEnvironment. Selection of derived type by get_environment() is not necessary in this case.
  References Dakota::abort_handler(), and Environment::environmentRep.

Environment ( ProgramOptions prog_opts )
Envelope constructor for LibraryEnvironment. Selection of derived type by get_environment() is not necessary in this case.
  References Dakota::abort_handler(), and Environment::environmentRep.

Environment ( MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts = ProgramOptions ( ) )
Envelope constructor for LibraryEnvironment. Selection of derived type by get_environment() is not necessary in this case.
  References Dakota::abort_handler(), and Environment::environmentRep.

Environment ( const String & env_type )
envelope constructor for letter type identified by String
  Alternate construction by String. Envelope constructor invokes get_environment() which instantiates a derived class letter; the derived constructor selects a BaseConstructor constructor in its initialization list to avoid the recursion of a base class constructor calling get_environment() again.
  References Dakota::abort_handler(), Environment::environmentRep, and Environment::get_environment().
Environment (const Environment & env)
copy constructor
  Copy constructor manages sharing of environmentRep and incrementing of referenceCount.
  References Environment::environmentRep, and Environment::referenceCount.

~Environment() [virtual]
destructor
  Destructor decrements referenceCount and only deletes environmentRep when referenceCount reaches zero.
  References Environment::destruct(), Environment::environmentRep, and Environment::referenceCount.

Environment (BaseConstructor) [protected]
constructor initializes the base class part of default-constructed letters
  This letter constructor initializes base class data for inherited environments that are default constructed. Since
  the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems
  in ~Environment).
  Use cases: library with no options, no MPI comm
  References ProgramOptions::exit_mode(), Environment::exit_mode(), WorkdirHelper::initialize(), and Environment-::programOptions.

Environment (BaseConstructor, int argc, char * argv[]) [protected]
constructor initializes the base class part of executable letters
  This letter constructor initializes base class data for inherited environments: instantiate/initialize the environ-
  ment, options, parallel library, and problem description database objects. Since the letter IS the representation, its
  representation pointer is set to NULL (an uninitialized pointer causes problems in ~Environment).
  Use cases: executable with command-line args
  References ProgramOptions::exit_mode(), Environment::exit_mode(), WorkdirHelper::initialize(), and Environment-::programOptions.

Environment (BaseConstructor, ProgramOptions prog_opts, MPI_Comm dakota_mpi_comm = MPI_COMM_WORLD) [protected]
constructor initializes the base class part of library letter classes
  This letter constructor initializes base class data for inherited environments. Since the letter IS the representa-
  tion, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Environment).
  Use cases: library with program options library with program options and MPI comm
  References ProgramOptions::exit_mode(), Environment::exit_mode(), WorkdirHelper::initialize(), and Environment-::programOptions.

13.49.3 Member Function Documentation
Environment operator= (const Environment & env)
assignment operator
  Assignment operator decrements referenceCount for old environmentRep, assigns new environmentRep, and
  References Environment::environmentRep, and Environment::referenceCount.
void exit_mode ( const String & mode = "exit" )

allow environment clients to set Dakota exit behavior (throw vs. exit)
Set the global variable controlling Dakota’s exit behavior. Call with no arguments to reset to default behavior.
References Dakota::abort_handler(), and Dakota::abort_mode.
Referenced by Environment::Environment(), and run_dakota_data().

void parse ( bool check_bcast_database = true, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL )

parse inputs, callbacks, and optionally check and broadcast
Parse input file and invoked any callbacks, then optionally check and sync database if check_bcast_database = true
References ProblemDescDB::check_and_broadcast(), ProgramOptions::input_file(), ProgramOptions::input_string(), ProblemDescDB::parse_inputs(), Environment::probDescDB, and Environment::programOptions.
Referenced by ExecutableEnvironment::ExecutableEnvironment(), and LibraryEnvironment::LibraryEnvironment().

Environment * get_environment ( const String & env_type ) [private]

Used by the envelope to instantiate the correct letter class.
Used only by the envelope constructor to initialize environmentRep to the appropriate derived type, as given by the environmentName attribute.
Referenced by Environment::Environment().
The documentation for this class was generated from the following files:

- DakotaEnvironment.hpp
- DakotaEnvironment.cpp

13.50  NomadOptimizer::Evaluator Class Reference

NOMAD-based Evaluator class.
Inherits Evaluator.

Public Member Functions

- Evaluator (const NOMAD::Parameters &p, Model &model)
  Constructor.
- ~Evaluator (void)
  Destructor.
- bool eval_x (NOMAD::EvalPoint &x, const NOMAD::Double &h_max, bool &count_eval) const
  Main Evaluation Method.
- void set_constraint_map (int numNomadNonlinearIneqConstraints, int numNomadNonlinearEqConstraints, std::vector<int> constraintMapIndices, std::vector<double> constraintMapMultipliers, std::vector<double> constraintMapOffsets)
  Publishes constraint transformation
- void set_surrogate_usage (std::string useSurrogate)
  Publishes surrogate usage
Private Attributes

- Model & _model
- int n_cont
- int n_disc_int
- int n_disc_real
- int numNomadNonlinearIneqConstr
  Number of nonlinear constraints after put into Nomad format.
- int numNomadNonlinearEqConstr
- std::vector<int> constrMapIndices
  map from Dakota constraint number to Nomad constraint number
- std::vector<double> constrMapMultipliers
  multipliers for constraint transformations
- std::vector<double> constrMapOffsets
  offsets for constraint transformations
- std::string useSgte
  defines use of surrogate in NOMAD

13.50.1 Detailed Description

NOMAD-based Evaluator class.

The NOMAD process requires an evaluation step, which calls the Simulation program. In the simplest version of this call, NOMAD executes the black box executable, which proceeds to write a file in a NOMAD-compatible format, which NOMAD reads to continue the process.

Because DAKOTA files are different form NOMAD files, and the simulations processed by DAKOTA already produce DAKOTA-compatible files, we cannot use this method for NOMAD. Instead, we implement the Nomad-Evaluator class, which takes the NOMAD inputs and passes them to DAKOTA's Interface for processing. The evaluator then passes the evaluation Responses into the NOMAD objects for further analysis.

13.50.2 Constructor & Destructor Documentation

Evaluator ( const NOMAD::Parameters & p, Model & model )

Constructor.

NOMAD Evaluator Constructor

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
</tr>
<tr>
<td>NOMAD Parameters object</td>
</tr>
<tr>
<td>model</td>
</tr>
<tr>
<td>DAKOTA Model object</td>
</tr>
</tbody>
</table>

13.50.3 Member Function Documentation

bool eval_x ( NOMAD::Eval_Point & x, const NOMAD::Double & h_max, bool & count_eval ) const

Main Evaluation Method.

Method that handles the communication between the NOMAD search process and the Black Box Evaluation managed by DAKOTA's Interface.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>Object that contains the points that need to be evaluated. Once the evaluation is completed, this object also stores the output back to be read by NOMAD.</td>
</tr>
<tr>
<td>( h_{\text{max}} )</td>
<td>Current value of the barrier parameter. Not used in this implementation.</td>
</tr>
<tr>
<td>( count_{\text{eval}} )</td>
<td>Flag that indicates whether this evaluation counts towards the max number of evaluations, often set to \text{false} when the evaluation does not meet certain costs during expensive evaluations. Not used in this implementation.</td>
</tr>
</tbody>
</table>

Returns

`true` if the evaluation was successful; `false` otherwise.

References Dakota::set_index_to_value().

The documentation for this class was generated from the following files:

- NomadOptimizer.hpp
- NomadOptimizer.cpp

13.51 JEGAOptimizer::Evaluator Class Reference

An evaluator specialization that knows how to interact with Dakota.

Inherits GeneticAlgorithmEvaluator.

Public Member Functions

- virtual bool Evaluate (DesignGroup &group)
  
  Does evaluation of each design in group.

- virtual bool Evaluate (Design &des)
  
  This method cannot be used!

- virtual std::string GetName () const
  
  Returns the proper name of this operator.

- virtual std::string GetDescription () const
  
  Returns a full description of what this operator does and how.

- virtual GeneticAlgorithmOperator * Clone (GeneticAlgorithm &algorithm) const
  
  Creates and returns a pointer to an exact duplicate of this operator.

- Evaluator (GeneticAlgorithm &algorithm, Model &model)

  Constructs a Evaluator for use by algorithm.

- Evaluator (const Evaluator &copy)

  Copy constructs a Evaluator.

- Evaluator (const Evaluator &copy, GeneticAlgorithm &algorithm, Model &model)

  Copy constructs a Evaluator for use by algorithm.

Static Public Member Functions

- static const std::string & Name ()
  
  Returns the proper name of this operator.

- static const std::string & Description ()
  
  Returns a full description of what this operator does and how.
13.51. **JEAOPTIMIZER::EVALUATOR CLASS REFERENCE**

**Protected Member Functions**

- void **SeparateVariables** (const Design &from, RealVector &intoCont, IntVector &intoDiscInt, RealVector &intoDiscReal, StringMultiArray &intoDiscString) const
  
  *This method fills intoCont, intoDiscInt and intoDiscReal appropriately using the values of from.*

- void **RecordResponses** (const RealVector &from, Design &into) const
  
  *Records the computed objective and constraint function values into into.*

- std::size_t **GetNumberNonLinearConstraints** () const
  
  *Returns the number of non-linear constraints for the problem.*

- std::size_t **GetNumberLinearConstraints** () const
  
  *Returns the number of linear constraints for the problem.*

**Private Member Functions**

- **Evaluator** (GeneticAlgorithm &algorithm)
  
  *This constructor has no implementation and cannot be used.*

**Private Attributes**

- Model & _model
  
  *The Model known by this evaluator.*

13.51.1 **Detailed Description**

An evaluator specialization that knows how to interact with Dakota. This evaluator knows how to use the model to do evaluations both in synchronous and asynchronous modes.

13.51.2 **Constructor & Destructor Documentation**

**Evaluator ( GeneticAlgorithm & algorithm, Model & model ) [inline]**

Constructs a **Evaluator** for use by **algorithm**.

The optimizer is needed for purposes of variable scaling.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
<tr>
<td>model</td>
<td>The model through which evaluations will be done.</td>
</tr>
</tbody>
</table>

**Evaluator ( const Evaluator & copy ) [inline]**

Copy constructs a **Evaluator**.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>The evaluator from which properties are to be duplicated into this.</td>
</tr>
</tbody>
</table>

**Evaluator ( const Evaluator & copy, GeneticAlgorithm & algorithm, Model & model ) [inline]**

Copy constructs a **Evaluator** for use by **algorithm**. The optimizer is needed for purposes of variable scaling.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>The existing Evaluator from which to retrieve properties.</td>
</tr>
<tr>
<td>algorithm</td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
<tr>
<td>model</td>
<td>The model through which evaluations will be done.</td>
</tr>
</tbody>
</table>

Evaluator ( GeneticAlgorithm & algorithm ) [private]

This constructor has no implementation and cannot be used.

This constructor can never be used. It is provided so that this operator can still be registered in an operator registry even though it can never be instantiated from there.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
</tbody>
</table>

13.51.3 Member Function Documentation

static const std::string& Name() [inline], [static]

Returns the proper name of this operator.

Returns

The string "DAKOTA JEGA Evaluator".

static const std::string& Description() [inline], [static]

Returns a full description of what this operator does and how.

The returned text is:

This evaluator uses Sandia’s DAKOTA optimization software to evaluate the passed in Designs. This makes it possible to take advantage of the fact that DAKOTA is designed to run on massively parallel machines.

Returns

A description of the operation of this operator.

void SeparateVariables ( const Design & from, RealVector & intoCont, IntVector & intoDiscInt, RealVector & intoDiscReal, StringMultiArray & intoDiscString ) const [protected]

This method fills intoCont, intoDiscInt and intoDiscReal appropriately using the values of from.

The discrete integer design variable values are placed in intoDiscInt, the discrete real design variable values are placed in intoDiscReal, and the continuum are placed into intoCont. The values are written into the vectors from the beginning so any previous contents of the vectors will be overwritten.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>The Design class object from which to extract the discrete design variable values.</td>
</tr>
<tr>
<td>intoDiscInt</td>
<td>The vector into which to place the extracted discrete integer values.</td>
</tr>
<tr>
<td>intoDiscReal</td>
<td>The vector into which to place the extracted discrete real values.</td>
</tr>
<tr>
<td>intoCont</td>
<td>The vector into which to place the extracted continuous values.</td>
</tr>
</tbody>
</table>

References JEGAOptimizer::Evaluator::model, Model::cv(), Model::discrete_int_sets(), Model::discrete_set_string_values(), Model::div(), Model::drv(), Model::dsv(), and Dakota::set_index_to_value().

void RecordResponses ( const RealVector & from, Design & into ) const [protected]

Records the computed objective and constraint function values into into.

This method takes the response values stored in from and properly transfers them into the into design.

The response vector from is expected to contain values for each objective function followed by values for each non-linear constraint in the order in which the info objects were loaded into the target by the optimizer class.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>The vector of responses to install into into.</td>
</tr>
<tr>
<td>into</td>
<td>The Design to which the responses belong and into which they must be written.</td>
</tr>
</tbody>
</table>

std::size_t GetNumberNonLinearConstraints ( ) const [inline], [protected]

Returns the number of non-linear constraints for the problem.

This is computed by adding the number of non-linear equality constraints to the number of non-linear inequality constraints. These values are obtained from the model.

Returns

The total number of non-linear constraints.

std::size_t GetNumberLinearConstraints ( ) const [inline], [protected]

Returns the number of linear constraints for the problem.

This is computed by adding the number of linear equality constraints to the number of linear inequality constraints. These values are obtained from the model.

Returns

The total number of linear constraints.

bool Evaluate ( DesignGroup & group ) [virtual]

Does evaluation of each design in group.

This method uses the Model known by this class to get Designs evaluated. It properly formats the Design class information in a way that Dakota will understand and then interprets the Dakota results and puts them back into the Design class object. It respects the asynchronous flag in the Model so evaluations may occur synchronously or asynchronously.

Prior to evaluating a Design, this class checks to see if it is marked as already evaluated. If it is, then the evaluation of that Design is not carried out. This is not strictly necessary because Dakota keeps track of evaluated designs and does not re-evaluate. An exception is the case of a population read in from a file complete with responses where Dakota is unaware of the evaluations.
Parameters

| group | The group of Design class objects to be evaluated. |

Returns

true if all evaluations completed and false otherwise.

**virtual bool Evaluate ( Design & des ) [inline], [virtual]**

This method cannot be used!!

This method does nothing and cannot be called. This is because in the case of asynchronous evaluation, this method would be unable to conform. It would require that each evaluation be done in a synchronous fashion.

Parameters

| des | A Design that would be evaluated if this method worked. |

Returns

Would return true if the Design were evaluated and false otherwise. Never actually returns here. Issues a fatal error. Otherwise, it would always return false.

**virtual std::string GetName ( ) const [inline], [virtual]**

Returns the proper name of this operator.

Returns

See Name().

**virtual std::string GetDescription ( ) const [inline], [virtual]**

Returns a full description of what this operator does and how.

Returns

See Description().

**virtual GeneticAlgorithmOperator* Clone ( GeneticAlgorithm & algorithm ) const [inline], [virtual]**

Creates and returns a pointer to an exact duplicate of this operator.

Parameters

| algorithm | The GA for which the clone is being created. |

Returns

A clone of this operator.
13.51.4 Member Data Documentation

Model& _model [private]

The Model known by this evaluator.
   It is through this model that evaluations will take place.
   Referenced by JEGAOptimizer::Evaluator::SeparateVariables().
   The documentation for this class was generated from the following file:

   • JEGAOptimizer.cpp

13.52 JEGAOptimizer::EvaluatorCreator Class Reference

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.
   Inherits EvaluatorCreator.

Public Member Functions

• virtual GeneticAlgorithmEvaluator* CreateEvaluator (GeneticAlgorithm &alg) [inline]
   Overriden to return a newly created Evaluator.
• EvaluatorCreator (Model &theModel) [inline]
   Constructs an EvaluatorCreator using the supplied model.

Private Attributes

• Model & _theModel
   The user defined model to be passed to the constructor of the Evaluator.

13.52.1 Detailed Description

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

13.52.2 Constructor & Destructor Documentation

EvaluatorCreator ( Model & theModel ) [inline]

Constructs an EvaluatorCreator using the supplied model.
Parameters

<table>
<thead>
<tr>
<th>theModel</th>
<th>The Dakota::Model this creator will pass to the created evaluator.</th>
</tr>
</thead>
</table>

13.52.3 Member Function Documentation

virtual GeneticAlgorithmEvaluator* CreateEvaluator ( GeneticAlgorithm & alg ) [inline], [virtual]

Overriden to return a newly created Evaluator.
   The GA will assume ownership of the evaluator so we needn’t worry about keeping track of it for destruction.
   The additional parameters needed by the Evaluator are stored as members of this class at construction time.
Parameters

| alg | The GA for which the evaluator is to be created. |

Returns

A pointer to a newly created Evaluator.

The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp

## 13.53 ExecutableEnvironment Class Reference

**Environment** corresponding to execution as a stand-alone application.

Inheritance diagram for ExecutableEnvironment:

```
Environment

ExecutableEnvironment
```

### Public Member Functions

- **ExecutableEnvironment ()**
  
  *default constructor*

- **ExecutableEnvironment (int argc, char ∗argv[ ∗])**
  
  *constructor*

- **∼ExecutableEnvironment ()**
  
  *destructor*

- **void execute ()**

  *the run function for the environment: invoke the iterator(s) on the model(s). Called from main.cpp.*

### Additional Inherited Members

#### 13.53.1 Detailed Description

**Environment** corresponding to execution as a stand-alone application.

This environment corresponds to a stand-alone executable program, e.g., main.cpp. It sets up the ParallelLibrary, ProgramOptions, and ProblemDescDB objects based on access to command line arguments.

The documentation for this class was generated from the following files:

- ExecutableEnvironment.hpp
- ExecutableEnvironment.cpp

## 13.54 ExperimentData Class Reference

Interpolation method for interpolating between experimental and model data. I need to work on inputs/outputs to this method. For now, this assumes interpolation of functional data.
Public Member Functions

- **ExperimentData ()**
  
  default constructor

- **ExperimentData (const ProblemDescDB &prob_desc_db, const SharedResponseData &srd, short output_level)**
  
  typical DB-based constructor

- **ExperimentData (size_t num_experiments, size_t num_config_vars, const boost::filesystem::path &data_prefix, const SharedResponseData &srd, const StringArray &variance_types, short output_level, std::string scalarDataFilename="")**
  
  temporary? constructor for testing

- **ExperimentData (size_t num_experiments, const SharedResponseData &srd, const RealMatrix &config-Vars, const IntResponseMap &all_responses, short output_level)**

- **void load_data (const std::string &context_message)**
  
  Load experiments from data files (simple scalar or field)

- **void add_data (const RealVector &one_configvars, const Response &one_response)**
  
  Add one data point to the experimental data set.

- **size_t num_experiments () const**
  
  retrieve the number of experiments

- **size_t num_total_exppoints () const**
  
  retrieve the total number of experimental data points over all experiments

- **size_t num_scalars () const**
  
  retrieve the number of scalars (applies to all experiments)

- **size_t num_fields () const**
  
  retrieve the number of fields (applies to all experiments)

- **size_t num_config_vars () const**
  
  number of configuration variables

- **const std::vector<RealVector> & config_vars () const**
  
  values of the configuration variables, 1 RealVector per experiment

- **const RealVector & all_data (size_t experiment)**
  
  return contiguous vector of all data (scalar, followed by field) for the specified experiment

- **const Response & response (size_t experiment)**
  
  return response for the specified experiment

- **void per_exp_length (IntVector &per_length) const**
  
  return the individual sizes of the experimental data lengths (all function values, scalar and field)

- **const IntVector & field_lengths (size_t experiment) const**
  
  return the field lengths for specified experiment index

- **Real scalar_data (size_t response, size_t experiment)***
  
  retrieve the data value for the given response, for the given experiment

- **RealVector field_data_view (size_t response, size_t experiment) const**
  
  retrieve a view of the field data for the given response, for the given experiment

- **RealMatrix field_coords_view (size_t response, size_t experiment) const**
  
  retrieve a view of the field data coordinates for the given response, for the given experiment

- **bool variance_type_active (short variance_type) const**
whether the specified variance type (enum value) is present and active

- bool variance_active () const
  whether any variance type is active

- Real apply_covariance (const RealVector &residuals, size_t experiment) const
  apply the covariance responses to compute the triple product \( v^\top \text{inv}(C)v \) for the given experiment

- void apply_covariance_inv_sqrt (const RealVector &residuals, size_t experiment, RealVector &weighted_residuals) const
  apply inverse \( \sqrt{\text{inv}} \) of the covariance to compute weighted residuals

- void apply_covariance_inv_sqrt (const RealMatrix &gradients, size_t experiment, RealMatrix &weighted_gradients) const
  apply inverse \( \sqrt{\text{inv}} \) of the covariance to compute weighted gradients

- void apply_covariance_inv_sqrt (const RealSymMatrixArray &hessians, size_t experiment, RealSymMatrixArray &weighted_hessians) const
  apply inverse \( \sqrt{\text{inv}} \) of the covariance to compute weighted Hessians

- void apply_simulation_error (const RealVector &simulation_error, size_t experiment)
  apply simulation error to experiment data

- void get_main_diagonal (RealVector &diagonal, size_t experiment) const
  return a (copy) vector containing the main diagonal entries of a specified experimental covariance matrix

- void cov_std_deviation (RealVectorArray &std_deviation) const
  get the standard deviation of the observation error process, one vector per experiment

- void cov_as_correlation (RealSymMatrixArray &corr_matrix) const
  get the observation error covariance as a correlation matrix, one vector per experiment

- void covariance (int exp_ind, RealSymMatrix &cov_mat) const
  retrieve an individual covariance entry as a dense matrix

- void form_residuals (const Response &sim_resp, Response &residual_resp) const
  form residuals for all experiments, interpolating if necessary; one simulation response maps to all experiments

- void form_residuals (const Response &sim_resp, const size_t curr_exp, Response &residual_resp) const
  Populate the portion of residual_resp corresponding to experiment curr_exp; the passed simulation response maps only to the specified experiment.

- void form_residuals (const Response &sim_resp, size_t exp_num, const ShortArray &total_asv, size_t residual_offset, Response &residual_resp) const
  form residuals for an individual experiment, interpolating if necessary

- void recover_model (size_t num_prfns, RealVector &model_fns) const
  recover original model from the first experiment block in a full set of residuals; works in no interpolation case only (sizes same)

- bool interpolate_flag () const
  flag for interpolation. If 0, no interpolation. If 1, interpolate.

- void interpolate_simulation_data (const Response &sim_resp, size_t exp_num, const ShortArray &total_asv, size_t exp_offset, Response &interpResp) const
  Interpolate simulation data (values, gradients and hessians) onto the coordinates of the experimental data.

- void scale_residuals (const Response &residual_response, RealVector &scaled_residuals) const
  Apply the experiment data covariance to the residual data (scale functions by \( \Gamma(d^{1/2}) \)), returning in scaled_residuals.

- void scale_residuals (Response &residual_response) const
Apply the experiment data covariance to the residual data in-place (scale functions, gradients, and Hessians by $\Gamma^{-1/2}$)

- void `build_gradient_of_sum_square_residuals` (const Response &resp, RealVector &ssr_gradient)
  
  Build the gradient of the ssr from residuals and function gradients based on the response’s active set request vector.

- void `build_gradient_of_sum_square_residuals` (const Response &resp, const ShortArray &asrv, RealVector &ssr_gradient)
  
  Build the gradient of the ssr from residuals and function gradients using the passed active set request vector (overrides the response’s request vector)

- void `build_gradient_of_sum_square_residuals_from_response` (const Response &resp, const ShortArray &asrv, int exp_ind, RealVector &ssr_gradient)
  
  Update the gradient of ssr with the values from the gradient associated with a single experiment.

- void `build_gradient_of_sum_square_residuals_from_function_data` (const RealMatrix &func_gradients, const RealVector &residuals, RealVector &ssr_gradient, const ShortArray &asrv)
  
  Construct the gradient of the sum of squares of residuals.

- void `build_hessian_of_sum_square_residuals` (const Response &resp, RealSymMatrix &ssr_hessian)
  
  Build the hessian of the ssr from residuals, function gradients and function hessians based on the response’s active set request vector.

- void `build_hessian_of_sum_square_residuals` (const Response &resp, const ShortArray &asrv, RealSymMatrix &ssr_hessian)
  
  Build the hessian of the ssr from residuals, function gradients and function hessians using the passed active set request vector (overrides the response’s request vector)

- void `build_hessian_of_sum_square_residuals_from_response` (const Response &resp, const ShortArray &asrv, int exp_ind, RealSymMatrix &ssr_hessian)
  
  Update the hessian of ssr with the values from the hessian associated with a single experiment.

- void `build_hessian_of_sum_square_residuals_from_function_data` (const RealSymMatrixArray &func_hessians, const RealMatrix &func_gradients, const RealVector &residuals, RealSymMatrix &ssr_hessian, const ShortArray &asrv)
  
  Construct the hessian of the sum of squares of residuals.

- void `scale_residuals` (const RealVector &multipliers, unsigned short multiplier_mode, size_t num_calib_params, Response &residual_response) const
  
  In-place scale the residual response (functions, gradients, Hessians) by $\sqrt{\text{multipliers}}$, according to blocks indicated by multiplier mode

- Real `cov_determinant` (const RealVector &multipliers, unsigned short multiplier_mode) const
  
  Returns the determinant of (covariance block-scaled by the passed multipliers)

- Real `half_log_cov_determinant` (const RealVector &multipliers, unsigned short multiplier_mode) const
  
  Returns the log of the determinant of (covariance block-scaled by the passed multipliers)

- void `half_log_cov_det_gradient` (const RealVector &multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealVector &gradient) const
  
  Populated the passed gradient with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- void `half_log_cov_det_hessian` (const RealVector &multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealSymMatrix &hessian) const
  
  Populated the passed Hessian with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- StringArray `hyperparam_labels` (unsigned short multiplier_mode) const
  
  Generate variable labels for the covariance (error) multiplier hyperparams
CHAPTER 13. CLASS DOCUMENTATION

Protected Member Functions

- ShortArray **determine_active_request** (const Response &resid_resp) const
  
  Perform check on the active request vector to make sure it is amenable to interpolation of simulation data and application of apply covariance.

- SizetArray **residuals_per_multiplier** (unsigned short multiplier_mode) const
  
  Count the number of residuals influenced by each multiplier.

- void **generate_multipliers** (const RealVector &multipliers, unsigned short multiplier_mode, RealVector &expanded_multipliers) const
  
  Generate a set of multipliers commensurate with the residual size for the total experiment data set. Instead of repeating the loops all over the place, generate an expanded set of multipliers; the conditionals get too complicated otherwise.

- void **resid2mult_map** (unsigned short multiplier_mode, IntVector &resid2mult_indices) const
  
  Return the index of the multiplier that affects each residual.

Private Member Functions

- void **initialize** (const StringArray &variance_types, const SharedResponseData &srd)
  
  Shared body of constructor initialization.

- void **parse_sigma_types** (const StringArray &sigma_types)
  
  Parse user-provided sigma type strings and populate enums.

- void **load_experiment** (size_t exp_index, std::ifstream &scalar_data_stream, size_t num_field_sigma_matrices, size_t num_field_sigma_diagonals, size_t num_field_sigma_scalars, size_t num_field_sigma_none, Response &exp_resp)
  
  Load a single experiment exp_index into exp_resp.

- void **read_scalar_sigma** (std::ifstream &scalar_data_stream, RealVector &sigma_scalars, IntVector &scalar_map_indices)
  
  Read or default populate the scalar sigma.

- RealVector **residuals_view** (const RealVector &residuals, size_t experiment) const
  
  Return a view (to allowing updating in place) of the residuals associated with a given experiment, from a vector containing residuals from all experiments.

- RealMatrix **gradients_view** (const RealMatrix &gradients, size_t experiment) const
  
  Return a view (to allowing updating in place) of the gradients associated with a given experiment, from a matrix containing gradients from all experiments.

- RealSymMatrixArray **hessians_view** (const RealSymMatrixArray &hessians, size_t experiment) const
  
  Return a view (to allowing updating in place) of the hessians associated with a given experiment, from an array containing the hessians from all experiments.

Private Attributes

- bool **calibrationDataFlag**
  
  Whether the user specified a calibration data block.

- size_t **numExperiments**
  
  The total number of experiments.

- size_t **numConfigVars**
  
  Number of configuration (state) variables to read for each experiment.
13.54. EXPERIMENTDATA CLASS REFERENCE

- **UShortArray varianceTypes**
  type of variance specified for each variable, one per response group; empty varianceType indicates none specified by user

- **Real covarianceDeterminant**
  cached product of each experiment covariance's determinant

- **Real logCovarianceDeterminant**
  cached sum of each experiment covariance's log determinant

- **boost::filesystem::path dataPathPrefix**
  path to prepend to any data file names

- **String scalarDataFilename**
  the user-specified scalar data filename

- **unsigned short scalarDataFormat**
  tabular format of the simple scalar data file; supports TABULAR_NONE, TABULAR_HEADER, TABULAR_EVAL_ID, TABULAR_EXPERANNOT

- **size_t scalarSigmaPerRow**
  number of sigma values to read from each row in simple data file format (calculated from variance types strings

- **bool readSimFieldCoords**
  whether to read coordinate data files for simulation fields

- **SharedPtrResponseData simulationSRD**
  archived shared data for use in sizing fields, total functions (historically we read all functions, including constraints, which might not be correct)

- **bool interpolateFlag**
  flag for interpolation.

- **short outputLevel**
  output verbosity level

- **std::vector< Response > allExperiments**
  Vector of numExperiments ExperimentResponses, holding the observed data and error (sigma/covariance) for each experiment.

- **std::vector< RealVector > allConfigVars**
  Vector of numExperiments configurations at which data were gathered; empty if no configurations specified.

- **IntVector experimentLengths**
  Length of each experiment.

- **IntVector expOffsets**
  function index offsets for individual experiment data sets

### 13.54.1 Detailed Description

Interpolation method for interpolating between experimental and model data. I need to work on inputs/outputs to this method. For now, this assumes interpolation of functional data.

As Brian suggested, this class has the experimental data (coordinates and RealVectorArray interpolatedResults; The ExperimentData class is used to read and populate data (currently from user-specified files and/or the input spec) relating to experimental (physical observations) data for the purposes of calibration. Such data may include (for example): number of experiments, configuration variables, type of data (scalar vs. functional), treatment of sigma (experimental uncertainties). This class also provides an interpolation capability to interpolate between simulation or experimental data so that the differencing between simulation and experimental data may be performed properly.
13.54.2 Member Function Documentation

`void form_residuals ( const Response & sim_resp, Response & residual_resp ) const`

form residuals for all experiments, interpolating if necessary; one simulation response maps to all experiments
This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.

References `ExperimentData::determine_active_request()`, `ExperimentData::numExperiments`, and ` ExperimentData::per_exp_length()`.

Referenced by `DataTransformModel::derived_evaluate()`, `ExperimentData::form_residuals()`, `DataTransformModel::primary_resp_differencer()`, and `DataTransformModel::transform_response_map()`.

`void form_residuals ( const Response & sim_resp, size_t exp_ind, const ShortArray & total_asv, size_t exp_offset, Response & residual_resp ) const`

form residuals for an individual experiment, interpolating if necessary
This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.

References `ExperimentData::allExperiments`, `ExperimentData::field_data_view()`, `Response::field_lengths()`, `Response::function_gradient_view()`, `Response::function_gradients()`, `Response::function_gradients_view()`, `Response::function_hessian_view()`, `Response::function_hessians()`, `Response::function_hessians_view()`, `Response::function_values()`, `Response::function_values_view()`, `ExperimentData::gradients_view()`, `ExperimentData::hessians_view()`, `ExperimentData::interpolate_simulation_data()`, `ExperimentData::interpolateFlag`, `ExperimentData::num_fields()`, `ExperimentData::num_scalars()`, and `ExperimentData::outputLevel()`.

`void recover_model ( size_t num_pri_fns, RealVector & best_fns ) const`

recover original model from the first experiment block in a full set of residuals; works in no interpolation case only (sizes same)
Add the data back to the residual to recover the model, for use in surrogated-based LSQ where DB lookup will fail (need approx eval DB). best_fns contains primary and secondary responses

References `Dakota::abort_handler()`, `ExperimentData::allExperiments`, `Response::function_value()`, `ExperimentData::interpolateFlag`, and `Response::num_functions()`.

`void build_gradient_of_sum_square_residuals_from_function_data ( const RealMatrix & func_gradients, const RealVector & residuals, RealVector & ssr_gradient, const ShortArray & asrv )`

Construct the gradient of the sum of squares of residuals.
Parameters

<table>
<thead>
<tr>
<th><code>func_gradients</code></th>
<th>A matrix containing the gradients of the residual vector</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>residuals</code></td>
<td>A vector of residuals (mismatch between experimental data and the corresponding function values)</td>
</tr>
<tr>
<td><code>asrv</code></td>
<td>The active set request vector</td>
</tr>
</tbody>
</table>

Referenced by `ExperimentData::build_gradient_of_sum_square_residuals_from_response()`.

`void build_hessian_of_sum_square_residuals_from_function_data ( const RealSymMatrixArray & func_hessians, const RealMatrix & func_gradients, const RealVector & residuals, RealSymMatrix & ssr_hessian, const ShortArray & asrv )`

Construct the hessian of the sum of squares of residuals.
13.54. EXPERIMENTDATA CLASS REFERENCE

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func_hessians</code></td>
<td>A list of matrices containing the Hessians of the function elements in the residual vector</td>
</tr>
<tr>
<td><code>func_gradients</code></td>
<td>A matrix containing the gradients of the residual vector</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>A vector of residuals (mismatch between experimental data and the corresponding function values</td>
</tr>
<tr>
<td><code>asrv</code></td>
<td>The active set request vector</td>
</tr>
</tbody>
</table>

Referenced by ExperimentData::build_hessian_of_sum_square_residuals_from_response().

```cpp
void scale_residuals ( const RealVector & multipliers, unsigned short multiplier_mode, size_t num_calib_params, Response & residual_response ) const
```

In-place scale the residual response (functions, gradients, Hessians) by sqrt(multipliers), according to blocks indicated by multiplier mode

- In-place scaling of residual response by hyper-parameter multipliers
  - References Dakota::abort_handler(), Response::active_set_request_vector(), Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), Response::function_value_view(), ExperimentData::num_total_exppoints(), and ExperimentData::resid2mult_map().

```cpp
Real cov_determinant ( const RealVector & multipliers, unsigned short multiplier_mode ) const
```

returns the determinant of (covariance block-scaled by the passed multipliers)

- Determinant of the total covariance used in inference, which has blocks mult_i * I * Cov_i.
  - References Dakota::abort_handler(), ExperimentData::covarianceDeterminant, ExperimentData::generate_multipliers(), and ExperimentData::num_total_exppoints().

```cpp
Real half_log_cov_determinant ( const RealVector & multipliers, unsigned short multiplier_mode ) const
```

returns the log of the determinant of (covariance block-scaled by the passed multipliers)

- Determinant of half the log of total covariance used in inference, which has blocks mult_i * I * Cov_i.
  - References Dakota::abort_handler(), ExperimentData::generate_multipliers(), ExperimentData::logCovarianceDeterminant, and ExperimentData::num_total_exppoints().
  - Referenced by NonDBayesCalibration::log_likelihood(), and NonDQUESOBayesCalibration::print_results().

```cpp
void half_log_cov_det_gradient ( const RealVector & multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealVector & gradient ) const
```

populated the passed gradient with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- Compute the gradient of scalar f(m) 0.5*log(det(mult*Cov)) w.r.t. mults. Since this is the only use case, we include the 0.5 factor and perform an update in-place.
  - References ExperimentData::num_total_exppoints(), and ExperimentData::residuals_per_multiplier().
  - Referenced by NonDBayesCalibration::neg_log_post_resp_mapping().

```cpp
void half_log_cov_det_hessian ( const RealVector & multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealSymMatrix & hessian ) const
```

populated the passed Hessian with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- Compute the gradient of scalar f(m) log(det(mult*Cov)) w.r.t. mults
  - References ExperimentData::num_total_exppoints(), and ExperimentData::residuals_per_multiplier().
CHAPTER 13. CLASS DOCUMENTATION

Referenced by NonDBayesCalibration::neg_log_post Resp_mapping().

SizetArray residuals_per_multiplier ( unsigned short multiplier_mode ) const [protected]

count the number of residuals influenced by each multiplier

Calculate how many residuals each multiplier affects

References ExperimentData::allExperiments, SharedResponseData::num_field_response_groups(), ExperimentData::num_fields(), SharedResponseData::num_response_groups(), SharedResponseData::num_scalar_responses(), ExperimentData::numExperiments, and ExperimentData::simulationSRD.

Referenced by ExperimentData::half_log_cov_det_gradient(), and ExperimentData::half_log_cov_det_hessian().

void parse_sigma_types ( const StringArray & sigma_types ) [private]

parse user-provided sigma type strings and populate enums

Validate user-provided sigma specification. User can specify 0, 1, or num_response_groups sigmas. If specified, sigma types must be the same for all scalar responses.

References Dakota::abort_handler(), SharedResponseData::num_response_groups(), SharedResponseData::num_scalar_responses(), ExperimentData::scalarDataFilename, ExperimentData::scalarSigmaPerRow, ExperimentData::simulationSRD, and ExperimentData::varianceTypes.

Referenced by ExperimentData::initialize().

void load_experiment ( size_t exp_index, std::ifstream & scalar_data_stream, size_t num_field_sigma_matrices, size_t num_field_sigma_diagonals, size_t num_field_sigma_scalars, size_t num_field_sigma_none, Response & exp_resp ) [private]

Load a single experiment exp_index into exp_resp.

Load an experiment from a mixture of legacy format data and field data format files

References Dakota::abort_handler(), ExperimentData::dataPathPrefix, Response::field_coords(), Response::field_group_labels(), ExperimentData::field_lengths(), Response::field_lengths(), Response::field_values(), Response::function_labels(), Response::function_value(), Dakota::is_matrix_symmetric(), SharedResponseData::num_field_response_groups(), ExperimentData::num_fields(), SharedResponseData::num_scalar_responses(), ExperimentData::num_scalars(), ExperimentData::read_scalar_sigma(), ExperimentData::scalarDataFilename, ExperimentData::scalarSigmaPerRow, Response::set_full_covariance(), ExperimentData::simulationSRD, and ExperimentData::varianceTypes.

Referenced by ExperimentData::load_data().

The documentation for this class was generated from the following files:

- ExperimentData.hpp
- ExperimentData.cpp

13.55 ExperimentResponse Class Reference

Container class for response functions and their derivatives. ExperimentResponse provides the body class.

Inheritance diagram for ExperimentResponse:

```
    Response
     |
     v
ExperimentResponse
```
Public Member Functions

- **ExperimentResponse ()**
  
  default constructor

- **ExperimentResponse (const Variables &vars, const ProblemDescDB &problem_db)**
  
  standard constructor built from problem description database

- **ExperimentResponse (const SharedResponseData &srd, const ActiveSet &set)**
  
  alternate constructor that shares a SharedResponseData instance

- **ExperimentResponse (const SharedResponseData &srd)**
  
  alternate constructor that shares a SharedResponseData instance

- **ExperimentResponse (const ActiveSet &set)**
  
  alternate constructor using limited data

- **~ExperimentResponse ()**
  
  destructor

- **void set_scalar_covariance (RealVector &scalars)**
  
  method to set the covariance matrix defined for ExperimentResponse

- **const ExperimentCovariance & experiment_covariance () const**
  
  retrieve the ExperimentCovariance structure

- **void set_full_covariance (std::vector<RealMatrix> &matrices, std::vector<RealVector> &diagonals, RealVector &scalars, IntVector matrix_map_indices, IntVector diagonal_map_indices, IntVector scalar_map_indices)**
  
  method to set the full covariance matrices for ExperimentResponse

- **Real apply_covariance (const RealVector &residual) const**
  
  method to compute the triple product $v' \ast \text{inv}(C) \ast v$.

- **void apply_covariance_inv_sqrt (const RealVector &residuals, RealVector &weighted_residuals) const**
  
  method to compute $(v' \ast \text{inv}(C) \ast v)^{-1/2}$, to compute weighted residual

- **void apply_covariance_inv_sqrt (const RealMatrix &gradients, RealMatrix &weighted_gradients) const**

- **void apply_covariance_inv_sqrt (const RealSymMatrixArray &hessians, RealSymMatrixArray &weighted_hessians) const**

- **void get_covariance_diagonal (RealVector &diagonal) const**

- **Real covariance_determinant () const**
  
  covariance determinant for this experiment (default 1.0)

- **Real log_covariance_determinant () const**
  
  log covariance determinant for this experiment (default 0.0)

Protected Member Functions

- **void copy_rep (Response *source_resp_rep)**
  
  Specialization of copy_rep; pulls base class data as well as derived specific data from the source rep into the this object.

Private Attributes

- **ExperimentCovariance expDataCovariance**
  
  sigma terms...
Additional Inherited Members

13.55.1 Detailed Description

Container class for response functions and their derivatives. ExperimentResponse provides the body class.

The ExperimentResponse class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (Response) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (ExperimentResponse) actually contains the response data (functionValues, functionGradients, functionHessians, etc.). The representation is hidden in that an instance of ExperimentResponse may only be created by Response. Therefore, programmers create instances of the Response handle class, and only need to be aware of the handle/body mechanisms when it comes to managing shallow copies (shared representation) versus deep copies (separate representation used for history mechanisms).

The documentation for this class was generated from the following files:

- ExperimentResponse.hpp
- ExperimentResponse.cpp

13.56 FileReadException Class Reference

base class for Dakota file read exceptions (to allow catching both tabular and general file truncation issues)

Inheritance diagram for FileReadException:

```
FileReadException

ResultsFileError TabularDataTruncated
```

Public Member Functions

- **FileReadException** (const std::string &msg)

13.56.1 Detailed Description

base class for Dakota file read exceptions (to allow catching both tabular and general file truncation issues)

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp

13.57 ForkApplicInterface Class Reference

Derived application interface class which spawns simulation codes using fork/execvp/waitpid.

Inheritance diagram for ForkApplicInterface:
Public Member Functions

- **ForkApplicInterface** (const ProblemDescDB &problem_db)
  - *constructor*
- **~ForkApplicInterface** ()
  - *destructor*

Protected Member Functions

- **void wait_local_evaluations** (PRPQueue &prp_queue)
  - For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.
- **void test_local_evaluations** (PRPQueue &prp_queue)
  - For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.
- **pid_t create_analysis_process** (bool block_flag, bool new_group)
  - spawn a child process for an analysis component within an evaluation using fork()/execvp() and wait for completion using waitpid() if block_flag is true
- **size_t wait_local_analyses** ()
  - wait for asynchronous analyses on the local processor, completing at least one job
- **size_t test_local_analyses_send** (int analysis_id)
  - test for asynchronous analysis completions on the local processor and return results for any completions by sending messages
- **void join_evaluation_process_group** (bool new_group)
  - create (if new group) and join the process group for async evaluations
- **void join_analysis_process_group** (bool new_group)
  - create (if new group) and join the process group for async analyses
- **void evaluation_process_group_id** (pid_t pgid)
  - set evalProcGroupIId
- **pid_t evaluation_process_group_id** () const
  - return evalProcGroupIId
- **void analysis_process_group_id** (pid_t pgid)
set analysisProcGroupId

- pid_t analysis_process_group_id () const
  
  return analysisProcGroupId

- pid_t wait_evaluation (bool block_flag)
  
  process all available completions within the evaluation process group; if block_flag = true, wait for at least one completion

- pid_t wait_analysis (bool block_flag)
  
  process all available completions within the analysis process group; if block_flag = true, wait for at least one completion

- void check_group (int err, pid_t proc_group_id)
  
  check the exit status of setpgid and abort if an error code was returned

Private Member Functions

- pid_t wait (pid_t proc_group_id, std::map< pid_t, int >&process_id_map, bool block_flag)
  
  core code used by wait_{evaluation,analysis}()

- void join_process_group (pid_t &process_group_id, bool new_group)
  
  core code used by join_{evaluation,analysis}_process_group()

Private Attributes

- pid_t evalProcGroupld
  
  the process group id used to identify a set of child evaluation processes used by this interface instance (to distinguish from other interface instances that could be running at the same time)

- pid_t analysisProcGroupld
  
  the process group id used to identify a set of child analysis processes used by this interface instance (to distinguish from other interface instances that could be running at the same time)

Additional Inherited Members

13.57.1 Detailed Description

Derived application interface class which spawns simulation codes using fork/execvp/waitpid. ForkApplicInterface is used on Unix systems and is a peer to SpawnApplicInterface for Windows systems. The documentation for this class was generated from the following files:

- ForkApplicInterface.hpp
- ForkApplicInterface.cpp

13.58 FSUDesignCompExp Class Reference

Wrapper class for the FSUDace QMC/CVT library. Inheritance diagram for FSUDesignCompExp:
Public Member Functions

- FSUDesignCompExp (ProblemDescDB &problem_db, Model &model)
  primary constructor for building a standard DACE iterator
- FSUDesignCompExp (Model &model, int samples, int seed, unsigned short sampling_method)
  alternate constructor for building a DACE iterator on-the-fly
- ~FSUDesignCompExp ()
  destructor
- bool resize ()
  reinitializes iterator based on new variable size

Protected Member Functions

- void pre_run ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- void post_input ()
  read tabular data for post-run mode
- void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- int num_samples () const
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator to use at least min_samples
- unsigned short sampling_scheme () const
  return sampling name
- void vary_pattern (bool pattern_flag)
  sets varyPattern in derived classes that support it
- void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
- void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix)
  Generate one block of numSamples samples (ndim * num_samples), populating design_matrix.
CHAPTER 13. CLASS DOCUMENTATION

Private Member Functions

- `void enforce_input_rules ()`
  enforce sanity checks/modifications for the user input specification

Private Attributes

- `int samplesSpec`
  initial specification of number of samples
- `int numSamples`
  current number of samples to be evaluated
- `bool allDataFlag`
  flag which triggers the update of allVars/allResponses for use by `Iterator::all_variables()` and `Iterator::all_responses()`
- `size_t numDACERuns`
  counter for number of executions for this object
- `bool latinizeFlag`
  flag which specifies latinization of QMC or CVT sample sets
- `IntVector sequenceStart`
  Integer vector defining a starting index into the sequence for random variable sampled. Default is 0 0 0 (e.g. for three random variables).
- `IntVector sequenceLeap`
  Integer vector defining the leap number for each sequence being generated. Default is 1 1 1 (e.g. for three random vars.)
- `IntVector primeBase`
  Integer vector defining the prime base for each sequence being generated. Default is 2 3 5 (e.g., for three random vars.)
- `int seedSpec`
  the user seed specification for the random number generator (allows repeatable results)
- `int randomSeed`
  current seed for the random number generator
- `bool varyPattern`
  flag for continuing the random number or QMC sequence from a previous execution (e.g., for surrogate-based optimization) so that multiple executions are repeatable but not identical.
- `int numCVTTrials`
  specifies the number of sample points taken at internal CVT iteration
- `int trialType`
  Trial type in CVT. Specifies where the points are placed for consideration relative to the centroids. Choices are grid (2), halton (1), uniform (0), or random (-1). Default is random.

Additional Inherited Members

13.58.1 Detailed Description

Wrapper class for the FSUDace QMC/CVT library.

The `FSUDesignCompExp` class provides a wrapper for FSUDace, a C++ design of experiments library from Florida State University. This class uses quasi Monte Carlo (QMC) and Centroidal Voronoi Tesselation (CVT) methods to uniformly sample the parameter space spanned by the active bounds of the current `Model`. It returns all generated samples and their corresponding responses as well as the best sample found.
13.58. FSUDESIGNCOMPEXP CLASS REFERENCE

13.58.2 Constructor & Destructor Documentation

FSUDesignCompExp ( ProblemDescDB & problem, Model & model )

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

References Dakota::abort_handler(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_iv(), ProblemDescDB::get_string(), Iterator::maxEvalConcurrency, Iterator::methodName, Analyzer::numContinuousVars, FSUDesignCompExp::numCVTTrials, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, Iterator::probDescDB, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, FSUDesignCompExp::trialType, and FSUDesignCompExp::varyPattern.

FSUDesignCompExp ( Model & model, int samples, int seed, unsigned short sampling_method )

alternate constructor for building a DACE iterator on-the-fly

This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.

References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Iterator::methodName, Analyzer::numContinuousVars, FSUDesignCompExp::numCVTTrials, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, and FSUDesignCompExp::trialType.

13.58.3 Member Function Documentation

void pre_run ( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Analyzer.

References FSUDesignCompExp::enforce_input_rules(), FSUDesignCompExp::get_parameter_sets(), Analyzer::get_vbd_parameter_sets(), Iterator::iteratedModel, FSUDesignCompExp::numSamples, Analyzer::pre_run(), and PStudyDACE::varBasedDecompFlag.

void core_run ( ) [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References FSUDesignCompExp::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, Analyzer::numLSqTerms, Analyzer::numObjFns, and Iterator::subIteratorFlag.

void post_run ( std::ostream & s ) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Analyzer::allResponses, Analyzer::allSamples, SensAnalysisGlobal::compute_correlations(), Analyzer::compute_vbd_stats(), FSUDesignCompExp::enforce_input_rules(), FSUDesignCompExp::numSamples, Analyzer::post_run(), PStudyDACE::pStudyDACESensGlobal, Iterator::subIteratorFlag, and PStudyDACE::varBasedDecompFlag.

```cpp
int num_samples() const [inline], [protected], [virtual]
```

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References FSUDesignCompExp::numSamples.

Referenced by FSUDesignCompExp::get parameter sets().

```cpp
void enforce_input_rules() [private]
```

enforce sanity checks/modifications for the user input specification

Users may input a variety of quantities, but this function must enforce any restrictions imposed by the sampling algorithms.

References Dakota::abort_handler(), Iterator::methodName, Analyzer::numContinuousVars, FSUDesignCompExp::numSamples, and FSUDesignCompExp::primeBase.

Referenced by FSUDesignCompExp::post_input(), FSUDesignCompExp::post_run(), and FSUDesignCompExp::pre_run().

The documentation for this class was generated from the following files:

- FSUDesignCompExp.hpp
- FSUDesignCompExp.cpp

### 13.59 FunctionEvalFailure Class Reference

exception class for function evaluation failures

Inherits runtime_error.

**Public Member Functions**

- `FunctionEvalFailure (const std::string &msg)`

#### 13.59.1 Detailed Description

exception class for function evaluation failures

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp
13.60 GaussProcApproximation Class Reference

Derived approximation class for Gaussian Process implementation.

Inheritance diagram for GaussProcApproximation:

```
Approximation
   GaussProcApproximation
```

### Public Member Functions

- **GaussProcApproximation()**
  - Default constructor
- **GaussProcApproximation(const SharedApproxData &shared_data)**
  - Alternate constructor
- **GaussProcApproximation(const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  - Standard constructor
- **~GaussProcApproximation()**
  - Destructor

### Protected Member Functions

- **int min_coefficients() const**
  - Return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions.
- **int num_constraints() const**
  - Return the number of constraints to be enforced via an anchor point.
- **void build(size_t index=NPOS)**
  - Find the covariance parameters governing the Gaussian process response.
- **Real value(const Variables &vars)**
  - Retrieve the function value for a given parameter set.
- **const RealVector &gradient(const Variables &vars)**
  - Retrieve the function gradient at the predicted value for a given parameter set.
- **Real prediction_variance(const Variables &vars)**
  - Retrieve the variance of the predicted value for a given parameter set.

### Private Member Functions

- **void GPmodel_build()**
  - Function to compute hyperparameters governing the GP.
- **void GPmodel_apply(const RealVector &new_x, bool variance_flag, bool gradients_flag)**
  - Function returns a response value using the GP surface.
• void normalize_training_data ()
  Normalizes the initial inputs upon which the GP surface is based.

• void get_trend ()
  Gets the trend (basis) functions for the calculation of the mean of the GP. If the order = 0, the trend is a constant, if the order = 1, trend is linear, if order = 2, trend is quadratic.

• void get_beta_coefficients ()
  Gets the beta coefficients for the calculation of the mean of the GP.

• int get_cholesky_factor ()
  Gets the Cholesky factorization of the covariance matrix, with error checking.

• void get_process_variance ()
  Gets the estimate of the process variance given the values of beta and the correlation lengthscales.

• void get_cov_matrix ()
  Calculates the covariance matrix for a given set of input points.

• void get_cov_vector ()
  Calculates the covariance vector between a new point x and the set of inputs upon which the GP is based.

• void optimize_theta_global ()
  Sets up and performs the optimization of the negative log likelihood to determine the optimal values of the covariance parameters using NCSUDirect.

• void optimize_theta_multipoint ()
  Sets up and performs the optimization of the negative log likelihood to determine the optimal values of the covariance parameters using a gradient-based solver and multiple starting points.

• void predict (bool variance_flag, bool gradients_flag)
  Calculates the predicted new response value for x in normalized space.

• Real calc_nll ()
  Calculates the negative log likelihood function (based on covariance matrix).

• void calc_grad_nll ()
  Gets the gradient of the negative log likelihood function with respect to the correlation lengthscales, theta.

• void get_grad_cov_vector ()
  Calculates the derivatives of the covariance vector, with respect to each component of x.

• void run_point_selection ()
  Runs the point selection algorithm, which will choose a subset of the training set with which to construct the GP model, and estimate the necessary parameters.

• void initialize_point_selection ()
  Initializes the point selection routine by choosing a small initial subset of the training points.

• void pointsel_get_errors (RealArray &delta)
  Uses the current GP model to compute predictions at all of the training points and find the errors.

• int addpoint (int, IntArray &added_index)
  Adds a point to the effective training set. Returns 1 on success.

• int pointsel_add_sel (const RealArray &delta)
  Accepts a vector of unsorted prediction errors, determines which points should be added to the effective training set, and adds them.

• Real maxval (const RealArray &)
  Returns the maximum value of the elements in a vector.

• void pointsel_write_points ()
13.60. GAUSSPROCAPPROXIMATION CLASS REFERENCE

- void **lhood_2d_grid_eval** ()
  
  *For problems with 2D input, evaluates the negative log likelihood on a grid.*

- void **writex** (const char[])
  
  *Writes out the current training set (in original units) to a specified file.*

- void **writeCovMat** (char[])
  
  *Writes out the covariance matrix to a specified file.*

**Static Private Member Functions**

- static void **negloglik** (int mode, int n, const Teuchos::SerialDenseVector<int, double> &X, Real &fx, Teuchos::SerialDenseVector<int, double> &grad_x, int &result_mode)
  
  *static function used by OPT++ as the objective function to optimize the hyperparameters in the covariance of the GP by minimizing the negative log likelihood*

- static void **constraint_eval** (int mode, int n, const Teuchos::SerialDenseVector<int, double> &X, Teuchos::SerialDenseVector<int, double> &g, Teuchos::SerialDenseMatrix<int, double> &gradC, int &result_mode)
  
  *static function used by OPT++ as the constraint function in the optimization of the negative log likelihood. Currently this function is empty: it is an unconstrained optimization.*

- static double **negloglikNCSU** (const RealVector &x)
  
  *function used by NCSUOptimizer to optimize negloglik objective*

**Private Attributes**

- Real **approxValue**
  
  *value of the approximation returned by value()*

- Real **approxVariance**
  
  *value of the approximation returned by prediction_variance()*

- RealMatrix **trainPoints**
  
  *A 2-D array (num sample sites = rows, num vars = columns) used to create the Gaussian process.*

- RealMatrix **trainValues**
  
  *An array of response values; one response value per sample site.*

- RealVector **trainMeans**
  
  *The mean of the input columns of trainPoints."

- RealVector **trainStdvs**
  
  *The standard deviation of the input columns of trainPoints.*

- RealMatrix **normTrainPoints**
  
  *Current working set of normalized points upon which the GP is based.*

- RealMatrix **trendFunction**
  
  *matrix to hold the trend function*

- RealMatrix **betaCoeffs**
  
  *matrix to hold the beta coefficients for the trend function*

- RealSymMatrix **covMatrix**
  
  *The covariance matrix where each element (i,j) is the covariance between points Xi and Xj in the initial set of samples.*
• RealMatrix covVector
  The covariance vector where each element (j,0) is the covariance between a new point X and point Xj from the initial set of samples.

• RealMatrix approxPoint
  Point at which a prediction is requested. This is currently a single point, but it could be generalized to be a vector of points.

• RealMatrix gradNegLogLikTheta
  matrix to hold the gradient of the negative log likelihood with respect to the theta correlation terms

• Teuchos::SerialSpdDenseSolver
  < int, Real > covSlvr
  The global solver for all computations involving the inverse of the covariance matrix.

• RealMatrix gradCovVector
  A matrix, where each column is the derivative of the covVector with respect to a particular component of X.

• RealMatrix normTrainPointsAll
  Set of all original samples available.

• RealMatrix trainValuesAll
  All original samples available.

• RealMatrix trendFunctionAll
  Trend function values corresponding to all original samples.

• RealMatrix Rinv_YFb
  Matrix for storing inverse of correlation matrix Rinv*(Y-FB)

• size_t numObs
  The number of observations on which the GP surface is built.

• size_t numObsAll
  The original number of observations.

• short trendOrder
  The number of variables in each X variable (number of dimensions of the problem).

• RealVector thetaParams
  Theta is the vector of covariance parameters for the GP. We determine the values of theta by optimization. Currently, the covariance function is theta[0]*exp(-0.5*sume)+delta*pow(sige,2). sume is the sum squared of weighted distances; it involves a sum of theta[1](Xi(1)-Xj(1))^2 + theta[2](Xi(2)-Xj(2))^2 + ... where Xi(1) is the first dimension value of multi-dimensional variable Xi. delta*pow(sige,2) is a jitter term used to improve matrix computations. delta is zero for the covariance between different points and 1 for the covariance between the same point. sige is the underlying process error.

• Real procVar
  The process variance, the multiplier of the correlation matrix.

• IntArray pointsAddedIndex
  Used by the point selection algorithm, this vector keeps track all points which have been added.

• int cholFlag
  A global indicator for success of the Cholesky factorization.

• bool usePointSelection
  A flag to indicate the use of point selection
13.61. **GENERALREADER CLASS REFERENCE**

**Static Private Attributes**
- static `GaussProcApproximation * GPinstance`  
  
  pointer to the active object instance used within the static evaluator

**Additional Inherited Members**

13.60.1 **Detailed Description**

Derived approximation class for Gaussian Process implementation.

The `GaussProcApproximation` class provides a global approximation (surrogate) based on a Gaussian process. The Gaussian process is built after normalizing the function values, with zero mean. Opt++ is used to determine the optimal values of the covariance parameters, those which minimize the negative log likelihood function.

13.60.2 **Constructor & Destructor Documentation**

`GaussProcApproximation()` [inline]

default constructor

alternate constructor used by EffGlobalOptimization and NonDGlobalReliability that does not use a problem database defaults here are no point selectinn and quadratic trend function.

13.60.3 **Member Function Documentation**

`void GPmodel_apply ( const RealVector & new_x, bool variance_flag, bool gradients_flag ) [private]`

Function returns a response value using the GP surface.

The response value is computed at the design point specified by the RealVector function argument.

References Dakota::abort_handler(), GaussProcApproximation::approxPoint, GaussProcApproximation::get_cov_vector(), SharedApproxData::numVars, GaussProcApproximation::predict(), Approximation::sharedDataRep, GaussProcApproximation::trainMeans, and GaussProcApproximation::trainStdvs.

Referenced by GaussProcApproximation::gradient(), GaussProcApproximation::pointsel_get_errors(), GaussProcApproximation::prediction_variance(), and GaussProcApproximation::value().

13.60.4 **Member Data Documentation**

`short trendOrder [private]`

The number of variables in each X variable (number of dimensions of the problem).

The order of the basis function for the mean of the GP If the order = 0, the trend is a constant, if the order = 1, trend is linear, if order = 2, trend is quadratic.

Referenced by GaussProcApproximation::GaussProcApproximation(), GaussProcApproximation::get_beta_coefficients(), GaussProcApproximation::get_trend(), GaussProcApproximation::GPmodel_build(), and GaussProcApproximation::predict().

The documentation for this class was generated from the following files:

- GaussProcApproximation.hpp
- GaussProcApproximation.cpp

**13.61 GeneralReader Class Reference**

Utility used in derived read_core to read in generic format.
Public Member Functions

- template<typename ArrayType>
  void operator() (std::istream &s, size_t start_index, size_t num_items, ArrayType &array_data, StringMultiArrayView label_array)

13.61.1 Detailed Description
Utility used in derived read_core to read in generic format.

The documentation for this class was generated from the following file:

- DakotaVariables.hpp

13.62 GeneralWriter Class Reference
Utility used in derived write_core to write in generic format.

Public Member Functions

- template<typename ArrayType>
  void operator() (std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)

13.62.1 Detailed Description
Utility used in derived write_core to write in generic format.

The documentation for this class was generated from the following file:

- DakotaVariables.hpp

13.63 GetLongOpt Class Reference
GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

Inheritance diagram for GetLongOpt:

```
GetLongOpt
  CommandLineHandler
```

Public Types

- enum OptType { Valueless, OptionalValue, MandatoryValue }

enum for different types of values associated with command line options.
Public Member Functions

- GetLongOpt (const char optmark= `-' )
  Constructor.
- ~GetLongOpt ()
  Destructor.
- int parse (int argc, char ∗const argv)
  parse the command line args (argc, argv).
- int parse (char ∗const str, char ∗const p)
  parse a string of options (typically given from the environment).
- int enroll (const char ∗const opt, const OptType t, const char ∗const desc, const char ∗const val)
  Add an option to the list of valid command options.
- const char ∗ retrieve (const char ∗const opt) const
  Retrieve value of option.
- void usage (std::ostream ∗const outfile=Cout) const
  Print usage information to outfile.
- void usage (const char ∗str)
  Change header of usage output to str.
- void store (const char ∗name, const char ∗value)
  Store a specified option value.

Private Member Functions

- char ∗ basename (char ∗const p) const
  extract the base name from a string as delimited by `/'
- int setcell (Cell ∗c, char ∗valtoken, char ∗nexttoken, const char ∗p)
  internal convenience function for setting Cell::value

Private Attributes

- Cell ∗ table
  option table
- const char ∗ ustring
  usage message
- char ∗ pname
  program basename
- char optmarker
  option marker
- int enroll_done
  finished enrolling
- Cell ∗ last
  last entry in option table
13.63.1 Detailed Description

GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

GetLongOpt manages the definition and parsing of "long options." Command line options can be abbreviated as long as there is no ambiguity. If an option requires a value, the value should be separated from the option either by whitespace or an "=".

13.63.2 Member Enumeration Documentation

enum OptType

enum for different types of values associated with command line options.

Enumerator

- **Valueless** option that may never have a value
- **OptionalValue** option with optional value
- **MandatoryValue** option with required value

13.63.3 Constructor & Destructor Documentation

**GetLongOpt** ( const char optmark = ' - ' )

Constructor.

Constructor for GetLongOpt takes an optional argument: the option marker. If unspecified, this defaults to ' - ', the standard (?) Unix option marker.


13.63.4 Member Function Documentation

**int parse** ( int argc, char *const argv )

parse the command line args (argc, argv).

A return value < 1 represents a parse error. Appropriate error messages are printed when errors are seen.

parse returns the the optind (see getopt(3)) if parsing is successful.


Referenced by CommandLineHandler::check_usage().

**int parse** ( char *const str, char *const p )

parse a string of options (typically given from the environment).

A return value < 1 represents a parse error. Appropriate error messages are printed when errors are seen.

parse takes two strings: the first one is the string to be parsed and the second one is a string to be prefixed to the parse errors.

int enroll ( const char *const opt, const OptType t, const char *const desc, const char *const val )

Add an option to the list of valid command options.

enroll adds option specifications to its internal database. The first argument is the option string. The second is an enum saying if the option is a flag (Valueless), if it requires a mandatory value (MandatoryValue) or if it takes an optional value (OptionalValue). The third argument is a string giving a brief description of the option. This description will be used by GetLongOpt::usage. GetLongOpt, for usage-printing, uses {$val} to represent values needed by the options. {$<$val>} is a mandatory value and {{[$val]}} is an optional value. The final argument to enroll is the default string to be returned if the option is not specified. For flags (options with Valueless), use "" (empty string, or in fact any arbitrary string) for specifying TRUE and 0 (null pointer) to specify FALSE.

References GetLongOpt::enroll_done, GetLongOpt::last, and GetLongOpt::table.

Referenced by CommandLineHandler::initialize_options().

const char * retrieve ( const char *const opt ) const

Retrieve value of option.

The values of the options that are enrolled in the database can be retrieved using retrieve. This returns a string and this string should be converted to whatever type you want. See atoi, atof, atol, etc. If a "parse" is not done before retrieving all you will get are the default values you gave while enrolling! Ambiguities while retrieving (may happen when options are abbreviated) are resolved by taking the matching option that was enrolled last. For example, -{v} will expand to -{-verify}. If you try to retrieve something you didn’t enroll, you will get a warning message.

References GetLongOpt::optmarker, and GetLongOpt::table.

Referenced by CommandLineHandler::check_usage(), ProgramOptions::manage_run_modes(), ProgramOptions::ProgramOptions(), and CommandLineHandler::read_restart_evals().

void usage ( const char * str ) [inline]

Change header of usage output to str.

GetLongOpt::usage is overloaded. If passed a string "str", it sets the internal usage string to "str". Otherwise it simply prints the command usage.

References GetLongOpt::ustring.

The documentation for this class was generated from the following files:

- CommandLineHandler.hpp
- CommandLineHandler.cpp

13.64 Graphics Class Reference

The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this OutputManager::dakotaGraphics.

Public Member Functions

- Graphics ()
  constructor
- ~Graphics ()
  destructor
- void createplots_2d (const Variables &vars, const Response &response)
creates the 2d graphics window and initializes the plots

- void add_datapoint (int graphics_cntr, const Variables &vars, const Response &response)
  adds data to each window in the 2d graphics based on the results of a model evaluation
- void add_datapoint (int i, double x, double y)
  adds data to a single window in the 2d graphics
- void new_dataset (int i)
  creates a separate line graphic for subsequent data points for a single window in the 2d graphics
- void close ()
  close graphics windows
- void set_x_labels2d (const char *x_label)
  set x label for each plot equal to x_label
- void set_y_labels2d (const char *y_label)
  set y label for each plot equal to y_label
- void set_x_label2d (int i, const char *x_label)
  set x label for ith plot equal to x_label
- void set_y_label2d (int i, const char *y_label)
  set y label for ith plot equal to y_label

Private Attributes

- Graphics2D * graphics2D
  pointer to the 2D graphics object
- bool win2dOn
  flag to indicate if 2D graphics window is active

13.64.1 Detailed Description

The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this OutputManager::dakotaGraphics.

13.64.2 Member Function Documentation

void create_plots_2d ( const Variables & vars, const Response & response )

creates the 2d graphics window and initializes the plots

Sets up a single event loop for duration of the dakotaGraphics object, continuously adding data to a single window. There is no reset. To start over with a new data set, you need a new object (delete old and instantiate new).

References Variables::continuous_variable_labels(), Variables::cv(), Variables::discrete_int_variable_labels(), Variables::discrete_real_variable_labels(), Variables::div(), Variables::drv(), Response::function_labels(), Graphics::graphics2D, Response::num_functions(), Dakota::re_match(), and Graphics::win2dOn.

Referenced by SurrBasedMinimizer::initialize_graphics(), NonDReliability::initialize_graphics(), and Iterator::initialize_graphics().
void add_datapoint ( int graphics_cntr, const Variables & vars, const Response & response )

void add_datapoint ( int i, double x, double y )

void new_dataset ( int i )

13.65 GridApplicInterface Class Reference

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.

Inheritance diagram for GridApplicInterface:

```
  Interface
  |________________|
  |                |
  V
ApplicationInterface
  |________________|
  |                |
  V
ProcessApplicInterface
  |________________|
  |                |
  V
SysCallApplicInterface
  |________________|
  |                |
  V
GridApplicInterface
```

The documentation for this class was generated from the following files:

- DakotaGraphics.hpp
- DakotaGraphics.cpp
Public Member Functions

- GridApplicInterface (const ProblemDescDB &problem_db)
  
  constructor
- ~GridApplicInterface ()
  
  destructor
- void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.
- void derived_map_asynch (const ParamResponsePair &pair)
  
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.
- void wait_local_evaluations (PRPQueue &prp_queue)
  
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.
- void test_local_evaluations (PRPQueue &prp_queue)
  
  Convenience function for common code between wait and nowait case.
- int synchronous_local_analysis (int analysis_id)

Protected Member Functions

- bool grid_file_test (const String &root_file)
  
  test file(s) for existence based on root_file name

Protected Attributes

- IntSet idSet
  
  Set of function evaluation id's for active asynchronous system call evaluations.
- IntShortMap failCountMap
  
  map linking function evaluation id's to number of response read failures
- start_grid_computing_t start_grid_computing
  
  handle to dynamically linked start_grid_computing function
- perform_analysis_t perform_analysis
  
  handle to dynamically linked perform_analysis grid function
- get_jobs_completed_t get_jobs_completed
  
  handle to dynamically linked get_jobs_completed grid function
- stop_grid_computing_t stop_grid_computing
  
  handle to dynamically linked stop_grid_computing function

13.65.1 Detailed Description

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus. This class is currently a modified copy of SysCallApplicInterface adapted for use with an external grid services library which was dynamically linked using dlopen() services.
13.65.2 Member Function Documentation

```cpp
int synchronous_local_analysis ( int analysis_id ) [inline], [virtual]
```

This code provides the derived function used by `ApplicationInterface::serve_analyses_synch()`. TODO - allow local analyses?????

Reimplemented from `ApplicationInterface`.

References `SysCallApplicInterface::spawn_analysis_to_shell()`.

The documentation for this class was generated from the following files:

- GridApplicInterface.hpp
- GridApplicInterface.cpp

13.66 HierarchSurrBasedLocalTraits Class Reference

Class for multilevel-multifidelity optimization algorithm.

Inheritance diagram for HierarchSurrBasedLocalTraits:

```
   TraitsBase
     |         |
     |         |
   HierarchSurrBasedLocalTraits
```

Public Member Functions

- `HierarchSurrBasedLocalTraits ()`
  
  default constructor

- `virtual ~HierarchSurrBasedLocalTraits ()`
  
  destructor

- `virtual bool is_derived ()`
  
  A temporary query used in the refactor.

- `bool supports_continuous_variables ()`
  
  Return the flag indicating whether method supports continuous variables.

- `bool supports_linear_equality ()`
  
  Return the flag indicating whether method supports linear equalities.

- `bool supports_linear_inequality ()`
  
  Return the flag indicating whether method supports linear inequalities.

- `bool supports_nonlinear_equality ()`
  
  Return the flag indicating whether method supports nonlinear equalities.

- `bool supports_nonlinear_inequality ()`
  
  Return the flag indicating whether method supports nonlinear inequalities.
13.66.1 Detailed Description

Class for multilevel-multifidelity optimization algorithm. This minimizer uses SurrogateModel(s) to perform minimization leveraging multiple model forms and discretization levels. A version of TraitsBase specialized for multilevel-multifidelity minimizer.

The documentation for this class was generated from the following file:

- HierarchSurrBasedLocalMinimizer.hpp

13.67 HierarchSurrModel Class Reference

Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

Inheritance diagram for HierarchSurrModel:

```
Model

SurrogateModel

HierarchSurrModel
```

Public Member Functions

- HierarchSurrModel (ProblemDescDB &problem_db)
  constructor
- ~HierarchSurrModel ()
  destructor
- const unsigned short correction_mode () const
  return correctionMode
- void correction_mode (unsigned short corr_mode)
  set correctionMode

Protected Member Functions

- DiscrepancyCorrection & discrepancy_correction ()
  return the DiscrepancyCorrection object used by SurrogateModels
- short correction_type ()
  return the correction type from the DiscrepancyCorrection object used by SurrogateModels
- bool initialize_mapping (ParLevLIter pl_iter)
  Perform any global updates prior to individual evaluate() calls.
- bool finalize_mapping ()
  restore state in preparation for next initialization
- void derived_evaluate (const ActiveSet &set)
  portion of evaluate() specific to HierarchSurrModel
• void derived_evaluate_nowait (const ActiveSet &set)
  portion of evaluate_nowait() specific to HierarchSurrModel
• const IntResponseMap & derived_synchronize ()
  portion of synchronize() specific to HierarchSurrModel
• const IntResponseMap & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to HierarchSurrModel
• Model & surrogate_model ()
  return the active low fidelity model
• void surrogate_model_indices (size_t lf_model_index, size_t lf_soln_lev_index=_NPOS)
  set the indices identifying the active low fidelity model
• void surrogate_model_indices (const SizetSizetPair &lf_form_level)
  set the index pair identifying the active low fidelity model
• const SizetSizetPair & surrogate_model_indices () const
  return the indices identifying the active low fidelity model
• SizetSizet2DPair get_indices ()
  return pair of active low fidelity and high fidelity model indices
• Model & truth_model ()
  return the active high fidelity model
• void truth_model_indices (size_t hf_model_index, size_t hf_soln_lev_index=_NPOS)
  set the indices identifying the active high fidelity model
• void truth_model_indices (const SizetSizetPair &hf_form_level)
  set the index pair identifying the active high fidelity model
• const SizetSizetPair & truth_model_indices () const
  return the indices identifying the active high fidelity model
• void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  return orderedModels and, optionally, their sub-model recursions
• void update_from_subordinate_model (size_t depth)
  update currentVariables using non-active data from the passed model (one of the ordered models)
• void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into LF/HF models
• void surrogate_response_mode (short mode)
  set responseMode and pass any bypass request on to the high fidelity model for any lower-level surrogate recursions
• void surrogate_function_indices (const IntSet &surr_fn_indices)
  (re)set the surrogate index set in SurrogateModel::surr_fnIndices
• void build_approximation (size_t index=_NPOS)
  use the high fidelity model to compute the truth values needed for correction of the low fidelity model results
• void component_parallel_mode (short mode)
  update component parallel mode for supporting parallelism in the low ad high fidelity models
• IntIntPair estimate_partition_bounds (int max_eval_concurrency)
  estimate the minimum and maximum partition sizes that can be utilized by this Model
• void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set up parallel operations for the array of ordered model fidelities
• **void derived_init_serial ()**
  set up serial operations for the array of ordered model fidelities

• **void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)**
  set active parallel configuration within the current low and high fidelity models identified by \{low,high\}Fidelity-Indices

• **void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)**
  deallocate communicator partitions for the HierarchSurrModel (request forwarded to the array of ordered model fidelities)

• **void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)**
  Service the low and high fidelity model job requests received from the master; completes when termination message received from stop_server().

• **void stop_servers ()**
  Executed by the master to terminate the low and high fidelity model server operations when iteration on the HierarchSurrModel is complete.

• **void inactive_view (short view, bool recurse_flag=true)**
  update the Model’s inactive view based on higher level (nested) context and optionally recurse into

• **bool evaluation_cache (bool recurse_flag=true) const**
  if recurse_flag, return true if orderedModels evaluation cache usage

• **bool restart_file (bool recurse_flag=true) const**
  if recurse_flag, return true if orderedModels restart file usage

• **void set_evaluation_reference ()**
  set the evaluation counter reference points for the HierarchSurrModel (request forwarded to the low and high fidelity models)

• **void fine_grained_evaluation_counters ()**
  request fine-grained evaluation reporting within the low and high fidelity models

• **void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const**
  print the evaluation summary for the HierarchSurrModel (request forwarded to the low and high fidelity models)

• **void warm_start_flag (const bool flag)**
  set the warm start flag, including the orderedModels

**Private Member Functions**

• **void check_interface_instance ()**
  update sameInterfaceInstance based on interface ids for models identified by current \{low,high\}FidelityIndices

• **void init_model (Model &model)**
  update the passed model (one of the ordered models) with data that could change once per set of evaluations (e.g., an outer iterator execution), including active variable labels, inactive variable values/bounds/labels, and linear/nonlinear constraint coeffs/bounds

• **void update_model (Model &model)**
  update the passed model (one of the ordered models) with data that could change per function evaluation (active variable values/bounds)

• **void update_from_model (Model &model)**
  update currentVariables using non-active data from the passed model (one of the ordered models)
• void derived\_synchronize\_sequential (IntResponseMap &hf\_resp\_map\_rekey, IntResponseMap &lf\_resp\_map\_rekey, bool block)
  called from derived\_synchronize\_nowait() to extract and rekey response maps using blocking or nonblocking synchronization on the LF and HF models

• void derived\_synchronize\_competing ()
  called from derived\_synchronize\_nowait() for case of distinct models/interfaces with competing LF/HF job queues

• void derived\_synchronize\_combine (const IntResponseMap &hf\_resp\_map, IntResponseMap &lf\_resp\_map, IntResponseMap &combined\_resp\_map)
  combine the HF and LF response maps into a combined response map

• void derived\_synchronize\_combine\_nowait (const IntResponseMap &hf\_resp\_map, IntResponseMap &lf\_resp\_map, IntResponseMap &combined\_resp\_map)
  combine the available components from HF and LF response maps into a combined response map

• void resize\_response ()
  resize currentResponse based on responseMode

• void compute\_apply\_delta (IntResponseMap &lf\_resp\_map)
  helper function used in the AUTO\_CORRECTED\_SURROGATE responseMode for computing a correction and applying it to lf\_resp\_map

• void single\_apply (const Variables &vars, Response &resp, const Sizet\_Sizet\_2DPair &indices)
  helper function for applying a single deltaCorr correction corresponding to indices

• void recursive\_apply (const Variables &vars, Response &resp)
  helper function for applying a correction across a sequence of model forms or discretization levels

• void stop\_model (size\_t ordered\_model\_index)
  stop the servers for the orderedModels instance identified by the passed index

Private Attributes

• DiscrepCorrMap deltaCorr
  manages construction and application of correction functions that are applied to a surrogate model (DataFitSurr or HierarchSurr) in order to reproduce high fidelity data.

• short corrOrder
  order of correction: 0, 1, or 2

• unsigned short correctionMode

• std::vector< Sizet\_Sizet\_2DPair > corrSequence
  vector to specify a sequence of discrepancy corrections to apply in AUTO\_CORRECTED\_SURROGATE mode

• ModelArray orderedModels
  Ordered sequence (low to high) of model fidelities. Models are of arbitrary type and supports recursions.

• Sizet\_SizetPair lowFidelityIndices
  index of the low fidelity model that is currently active within orderedModels; provides approximate low fidelity function evaluations.

• Sizet\_SizetPair highFidelityIndices
  index of the high fidelity model that is currently active within orderedModels; provides truth evaluations for computing corrections to the low fidelity results.

• bool sameModelInstance
  flag indicating that the \{low,high\}FidelityIndices correspond to the same model instance, requiring modifications to updating and evaluation scheduling processes
• `bool sameInterfaceInstance`

  flag indicating that the models identified by `{low, high}` `FidelityIndices` employ the same interface instance, requiring modifications to evaluation scheduling processes

• `SizetSizetPair componentParallelIndices`

  store index pair that is active in `component_parallel_mode()`

• `std::map<SizetSizetPair, Response> truthResponseRef`

  map of reference truth (high fidelity) responses computed in `build_approximation()` and used for calculating corrections

• `IntResponseMap cachedTruthRespMap`

  map of truth (high-fidelity) responses retrieved in `derived_synchronize_nowait()` that could not be returned since corresponding low-fidelity response portions were still pending

### Additional Inherited Members

13.67.1 Detailed Description

Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

The `HierarchSurrModel` class manages hierarchical models of varying fidelity. The class contains an ordered array of model forms (fidelity ordered from low to high), where each model form may also contain a set of solution levels (space/time discretization, convergence tolerances, etc.). At run time, one of these combinations is activated as the low fidelity model and used to perform approximate function evaluations, while another of these combinations is activated as the high fidelity model and used to provide truth evaluations for computing corrections to the low fidelity results.

13.67.2 Member Function Documentation

`bool initialize_mapping ( ParLevLIter pl_iter )` [protected], [virtual]

Perform any global updates prior to individual `evaluate()` calls.

Inactive variables must be propagated when a `HierarchSurrModel` is employed by a sub-iterator (e.g., OUU with MLMC or MLPCE). In current use cases, this can occur once per sub-iterator execution within `Model::initialize_mapping()`.

Reimplemented from `Model`.

References `HierarchSurrModel::init_model()`, `SurrogateModel::initialize_mapping()`, and `HierarchSurrModel::orderedModels`.

`bool finalize_mapping ( )` [protected], [virtual]

restore state in preparation for next initialization

Inactive variables must be propagated when a `HierarchSurrModel` is employed by a sub-iterator (e.g., OUU with MLMC or MLPCE). In current use cases, this can occur once per sub-iterator execution within `Model::initialize_mapping()`.

Reimplemented from `Model`.

References `SurrogateModel::finalize_mapping()`, and `HierarchSurrModel::orderedModels`. 
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]

portion of evaluate() specific to HierarchSurrModel

  Compute the response synchronously using LF model, HF model, or both (mixed case). For the LF model portion, compute the high fidelity response if needed with build_approximation(), and, if correction is active, correct the low fidelity results.

  Reimplemented from Model.

  References Response::active_set(), SurrogateModel::aggregate_response(), SurrogateModel::approxBuilds, SurrogateModel::asv_mapping(), HierarchSurrModel::build_approximation(), HierarchSurrModel::component_parallel_mode(), Response::copy(), Model::current_response(), Model::currentResponse, Model::currentVariables, HierarchSurrModel::deltaCorr, Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate(), SurrogateModel::force_rebuild(), HierarchSurrModel::get_indices(), Model::hierarchicalTagging, HierarchSurrModel::highFidelityIndices, HierarchSurrModel::lowFidelityIndices, HierarchSurrModel::orderedModels, Model::outputLevel, HierarchSurrModel::recursive_apply(), ActiveSet::request_vector(), SurrogateModel::response_mapping(), SurrogateModel::responseMode, HierarchSurrModel::s conducts, Model::solution_level_index(), SurrogateModel::surrModelEvalCntr, Response::update(), and HierarchSurrModel::update_model().

void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]

portion of evaluate_nowait() specific to HierarchSurrModel

  Compute the response asynchronously using LF model, HF model, or both (mixed case). For the LF model portion, compute the high fidelity response with build_approximation() (for correcting the low fidelity results in derived_synchronize() and derived_synchronize_nowait()) if not performed previously.

  Reimplemented from Model.

  References SurrogateModel::approxBuilds, SurrogateModel::asv_mapping(), Model::asynch_flag(), HierarchSurrModel::build_approximation(), SurrogateModel::cachedApproxRespMap, HierarchSurrModel::cachedTruthRespMap, HierarchSurrModel::component_parallel_mode(), Response::copy(), Variables::copy(), Model::current_response(), Model::currentVariables, ActiveSet::derivative_vector(), Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate(), Model::evaluate_nowait(), Model::evaluation_id(), SurrogateModel::force_rebuild(), Model::hierarchicalTagging, HierarchSurrModel::highFidelityIndices, HierarchSurrModel::lowFidelityIndices, HierarchSurrModel::orderedModels, SurrogateModel::rawVarsMap, HierarchSurrModel::recursive_apply(), ActiveSet::request_vector(), SurrogateModel::responseMode, HierarchSurrModel::sameModelInstance, Model::solution_level_index(), SurrogateModel::surrIdMap, SurrogateModel::surrModelEvalCntr, SurrogateModel::truthIdMap, and HierarchSurrModel::update_model().

const IntResponseMap & derived_synchronize ( ) [protected], [virtual]

portion of synchronize() specific to HierarchSurrModel

  Blocking retrieval of asynchronous evaluations from LF model, HF model, or both (mixed case). For the LF model portion, apply correction (if active) to each response in the array. derived_synchronize() is designed for the general case where derived_evaluate_nowait() may be inconsistent in its use of low fidelity evaluations, high fidelity evaluations, or both.

  Reimplemented from Model.

  References HierarchSurrModel::derived_synchronize_combine(), HierarchSurrModel::derived_synchronize_competing(), HierarchSurrModel::derived_synchronize_sequential(), HierarchSurrModel::sameInterfaceInstance, HierarchSurrModel::sameModelInstance, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, and SurrogateModel::truthIdMap.

const IntResponseMap & derived_synchronize_nowait ( ) [protected], [virtual]

portion of synchronize_nowait() specific to HierarchSurrModel
Nonblocking retrieval of asynchronous evaluations from LF model, HF model, or both (mixed case). For the LF model portion, apply correction (if active) to each response in the map. `derived_synchronize_nowait()` is designed for the general case where `derived_evaluate_nowait()` may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from `Model`.

References `HierarchSurrModel::derived_synchronize_combine_nowait()`, `HierarchSurrModel::derived_synchronize_sequential()`, and `SurrogateModel::surrResponseMap`.

Referenced by `HierarchSurrModel::derived_synchronize_competing()`.

The documentation for this class was generated from the following files:

- `HierarchSurrModel.hpp`
- `HierarchSurrModel.cpp`

### 13.68 Interface Class Reference

Base class for the interface class hierarchy.

Inheritance diagram for `Interface`:

```
  Interface
    |      |      |
    +-----+-----+
    | ApplicationInterface | ApproximationInterface |
    |                  |                        |
    +-----------------+---------------------+
    | DirectApplicInterface | ProcessApplicInterface |
    |                  |                        |
    +-----------------+---------------------+
    | MatlabInterface  | ProcessHandleApplicInterface |
    |                  |                        |
    +-----------------+---------------------+
    | PythonInterface | SysCallApplicInterface |
    |                  |                        |
    +-----------------+---------------------+
    | ScilabInterface |
    |                  |
    +-----------------+
    | TestDriverInterface |
    |                  |
    +-----------------+
    | ParallelDirectApplicInterface |
    |                  |
    +-----------------+
    | SerialDirectApplicInterface |
```

#### Public Member Functions

- **`Interface()`**
  
  *default constructor*

- **`Interface (ProblemDescDB &problem_db)`**
  
  *standard constructor for envelope*

- **`Interface (const Interface &interface_in)`**
  
  *copy constructor*

- **virtual `~Interface()`**
  
  *destructor*

- **`Interface operator= (const Interface &interface_in)`**
  
  *assignment operator*
• virtual void **map** (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)
  the function evaluator: provides a "mapping" from the variables to the responses.

• virtual const IntResponseMap & **synchronize** ()
  recovers data from a series of asynchronous evaluations (blocking)

• virtual const IntResponseMap & **synchronize_nowait** ()
  recovers data from a series of asynchronous evaluations (nonblocking)

• virtual void **serve_evaluations** ()
  evaluation server function for multiprocessor executions

• virtual void **stop_evaluation_servers** ()
  send messages from iterator rank 0 to terminate evaluation servers

• virtual void **init_communicators** (const IntArray &message_lengths, int max_eval_concurrency)
  allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

• virtual void **set_communicators** (const IntArray &message_lengths, int max_eval_concurrency)
  set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

• virtual void **init_serial** ()
  reset certain defaults for serial interface objects.

• virtual int **asynch_local_evaluation_concurrency** () const
  return the user-specified concurrency for asynch local evaluations

• virtual short **interface_synchronization** () const
  return the user-specified interface synchronization

• virtual int **minimum_points** (bool constraint_flag) const
  returns the minimum number of points required to build a particular ApproximationInterface (used by DataFitSurrModels).

• virtual int **recommended_points** (bool constraint_flag) const
  returns the recommended number of points required to build a particular ApproximationInterface (used by DataFitSurrModels).

• virtual void **approximation_function_indices** (const IntSet &approx_fn_indices)
  set the (currently active) approximation function index set

• virtual void **update_approximation** (const Variables &vars, const IntResponsePair &response_pr)
  updates the anchor point for an approximation

• virtual void **update_approximation** (const RealMatrix &samples, const IntResponseMap &resp_map)
  updates the current data points for an approximation

• virtual void **update_approximation** (const VariablesArray &vars_array, const IntResponseMap &resp_map)
  updates the current data points for an approximation

• virtual void **append_approximation** (const Variables &vars, const IntResponsePair &response_pr)
  appends a single point to an existing approximation

• virtual void **append_approximation** (const RealMatrix &samples, const IntResponseMap &resp_map)
  appends multiple points to an existing approximation

• virtual void **append_approximation** (const VariablesArray &vars_array, const IntResponseMap &resp_map)
  appends multiple points to an existing approximation

• virtual void **build_approximation** (const RealVector &c_lbnds, const RealVector &c_ubnds, const IntVector &di_l_bnds, const IntVector &di_u_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds, size_t index=NPOS)
CHAPTER 13. CLASS DOCUMENTATION

- builds the approximation
- virtual void `export_approximation()`
  export the approximation to disk
- virtual void `rebuild_approximation` (const BoolDeque &rebuild_deque, size_t index=_NPOS)
  rebuilds the approximation after a data update
- virtual void `pop_approximation` (bool save_surrogate_data)
  removes data from last append from the approximation
- virtual void `push_approximation`()
  retrieves approximation data from a previous state (negates pop)
- virtual bool `push_available`()
  queries the approximation for the ability to retrieve a previous increment
- virtual void `finalize_approximation`()
  finalizes the approximation by applying all trial increments
- virtual void `store_approximation` (size_t index=_NPOS)
  move the current approximation into storage for later combination; the index of the stored approximation can be passed to allow replacement instead of augmentation (default is push_back)
- virtual void `restore_approximation` (size_t index=_NPOS)
  return an approximation from storage; the index identifies a particular stored data set (default is pop_back from stored)
- virtual void `remove_stored_approximation` (size_t index=_NPOS)
  remove a stored approximation, due to redundancy with the current approximation, prior to combination (default for no index is pop_back from stored)
- virtual void `combine_approximation`()
  combine the current approximation with previously stored data sets
- virtual void `clear_stored`()
  clear stored approximation data
- virtual `Real2DArray cv_diagnostics` (const StringArray &metric_types, unsigned num_folds)
  approximation cross-validation quality metrics per response function
- virtual `RealArray challenge_diagnostics` (const String &metric_type, const RealMatrix &challenge_pts)
  approximation challenge data metrics per response function
- virtual void `clear_current`()
  clears current data from an approximation interface
- virtual void `clear_all`()
  clears all data from an approximation interface
- virtual void `clear_popped`()
  clears bookkeeping for popped data sets from an approximation interface
- virtual `SharedApproxData & shared_approximation`()
  retrieve the `SharedApproxData` within an `ApproximationInterface`
- virtual std::vector< Approximation > & approximations() 
  retrieve the Approximations within an `ApproximationInterface`
- virtual const
  Pecos::SurrogateData & `approximation_data` (size_t index)
  retrieve the approximation data from a particular `Approximation` within an `ApproximationInterface`
virtual const RealVectorArray & approximation_coefficients (bool normalized=false)
  retrieve the approximation coefficients from each Approximation within an ApproximationInterface

virtual void approximation_coefficients (const RealVectorArray &approx_coeffs, bool normalized=false)
  set the approximation coefficients within each Approximation within an ApproximationInterface

virtual const RealVector & approximation_variances (const Variables &vars)
  retrieve the approximation variances from each Approximation within an ApproximationInterface

virtual const StringArray & analysis_drivers () const
  retrieve the analysis drivers specification for application interfaces

virtual bool evaluation_cache () const
  return flag indicating usage of the global evaluation cache

virtual bool restart_file () const
  return flag indicating usage of the restart file

virtual void file_cleanup () const
  clean up any interface parameter/response files when aborting

void cache_unmatched_response (int raw_id)
  migrate an unmatched response record from rawResponseMap to cachedResponseMap

void assign_rep (Interface *interface_rep, bool ref_count_incr=true)
  assign letter or replace existing letter with a new one

unsigned short interface_type () const
  returns the interface type

const String & interface_id () const
  returns the interface identifier

int evaluation_id () const
  returns the value of the (total) evaluation id counter for the interface

void fine_grained_evaluation_counters (size_t num_fns)
  set fineGrainEvalCounters to true and initialize counters if needed

void init_evaluation_counters (size_t num_fns)
  initialize fine grained evaluation counters, sizing if needed

void set_evaluation_reference ()
  set evaluation count reference points for the interface

void print_evaluation_summary (std::ostream &s, bool minimal_header, bool relative_count) const
  print an evaluation summary for the interface

bool multi_proc_eval () const
  returns a flag signaling the use of multiprocessor evaluation partitions

bool iterator_eval_dedicated_master () const
  returns a flag signaling the use of a dedicated master processor at the iterator-evaluation scheduling level

bool is_null () const
  function to check interfaceRep (does this envelope contain a letter?)

void eval_tag_prefix (const String &eval_id_str, bool append_iface_id=true)
  set the evaluation tag prefix (does not recurse)
Protected Member Functions

- **Interface** (BaseConstructor, const ProblemDescDB &problem_db)
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Interface** (NoDBBaseConstructor, size_t num_fns, short output_level)
  
  constructor initializes the base class part of letter classes (NoDBBaseConstructor used for on the fly instantiations without a DB)

- void **init_algebraic_mappings** (const Variables &vars, const Response &response)
  
  Define algebraicACVIndices, algebraicACVIds, and algebraicFnIndices.

- void **asv_mapping** (const ActiveSet &total_set, ActiveSet &algebraic_set, ActiveSet &core_set)
  
  define the evaluation requirements for algebraic_mappings() (algebraic_set) and the core Application/Approximation mapping (core_set) from the total Interface evaluation requirements (total_set)

- void **asv_mapping** (const ActiveSet &algebraic_set, ActiveSet &total_set)
  
  map an algebraic ASV back to original total ordering for asynch recovery

- void **algebraic_mappings** (const Variables &vars, const ActiveSet &algebraic_set, Response &algebraic_response)
  
  evaluate the algebraic_response using the AMPL solver library and the data extracted from the algebraic_mappings file

- void **response_mapping** (const Response &algebraic_response, const Response &core_response, Response &total_response)
  
  combine the response from algebraic_mappings() with the response from derived_map() to create the total response

- String **final_eval_id_tag** (int fn_eval_id)
  
  form and return the final evaluation ID tag, appending iface ID if needed

Protected Attributes

- unsigned short **interfaceType**
  
  the interface type: enum for system, fork, direct, grid, or approximation

- String **interfaceId**
  
  the interface specification identifier string from the DAKOTA input file

- bool **algebraicMappings**
  
  flag for the presence of algebraic_mappings that define the subset of an Interface’s parameter to response mapping that is explicit and algebraic.

- bool **coreMappings**
  
  flag for the presence of non-algebraic mappings that define the core of an Interface’s parameter to response mapping (using analysis_drivers for ApplicationInterface or functionSurfaces for ApproximationInterface).

- short **outputLevel**
  
  output verbosity level: {SILENT,QUIET,NORMAL,VERBOSE,DEBUG}_OUTPUT

- int **currEvalId**
  
  identifier for the current evaluation, which may differ from the evaluation counters in the case of evaluation scheduling; used on iterator master as well as server processors. Currently, this is set prior to all invocations of derived_map() for all processors.

- bool **fineGrainEvalCounters**
  
  controls use of fn val/grad/hess counters for detailed evaluation report

- int **evalIdCntr**
- int newEvalIdCntr
  new (non-duplicate) interface evaluation counter
- int evalIdRefPt
  iteration reference point for evalIdCntr
- int newEvalIdRefPt
  iteration reference point for newEvalIdCntr
- IntArray fnValCounter
  number of value evaluations by resp fn
- IntArray fnGradCounter
  number of gradient evaluations by resp fn
- IntArray fnHessCounter
  number of Hessian evaluations by resp fn
- IntArray newFnValCounter
  number of new value evaluations by resp fn
- IntArray newFnGradCounter
  number of new gradient evaluations by resp fn
- IntArray newFnHessCounter
  number of new Hessian evaluations by resp fn
- IntArray fnValRefPt
  iteration reference point for fnValCounter
- IntArray fnGradRefPt
  iteration reference point for fnGradCounter
- IntArray fnHessRefPt
  iteration reference point for fnHessCounter
- IntArray newFnValRefPt
  iteration reference point for newFnValCounter
- IntArray newFnGradRefPt
  iteration reference point for newFnGradCounter
- IntArray newFnHessRefPt
  iteration reference point for newFnHessCounter
- IntResponseMap rawResponseMap
  Set of responses returned by either a blocking or nonblocking schedule.
- IntResponseMap cachedResponseMap
  Set of available asynchronous responses completed within a blocking or nonblocking scheduler that cannot be processed in a higher level context and need to be stored for later.
- StringArray fnLabels
  response function descriptors (used in print_evaluation_summary() and derived direct interface classes); initialized in map() functions due to potential updates after construction
- bool multiProcEvalFlag
  flag for multiprocessor evaluation partitions (evalComm)
- bool ieDedMasterFlag
  flag for dedicated master partitioning at the iterator level
• String evalTagPrefix
  set of period-delimited evaluation ID tags to use in evaluation tagging
• bool appendIfaceId
  whether to append the interface ID to the prefix during map (default true)

Private Member Functions

• Interface * get_interface (ProblemDescDB &problem_db)
  Used by the envelope to instantiate the correct letter class.
• int algebraic_function_type (String)
  Used by algebraic mappings to determine the correct AMPL function evaluation call to make.

Private Attributes

• StringArray algebraicVarTags
  set of variable tags from AMPL stub.col
• SizetArray algebraicACVIndices
  set of indices mapping AMPL algebraic variables to DAKOTA all continuous variables
• SizetArray algebraicACVIds
  set of ids mapping AMPL algebraic variables to DAKOTA all continuous variables
• StringArray algebraicFnTags
  set of function tags from AMPL stub.row
• IntArray algebraicFnTypes
  function type: > 0 = objective, < 0 = constraint |value|-1 is the objective (constraint) index when making AMPL objval (conival) calls
• SizetArray algebraicFnIndices
  set of indices mapping AMPL algebraic objective functions to DAKOTA response functions
• RealArray algebraicConstraintWeights
  set of weights for computing Hessian matrices for algebraic constraints;
• int numAlgebraicResponses
  number of algebraic responses (objectives+constraints)
• Interface * interfaceRep
  pointer to the letter (initialized only for the envelope)
• int referenceCount
  number of objects sharing interfaceRep
• ASL * asl
  pointer to an AMPL solver library (ASL) object

13.68.1 Detailed Description

Base class for the interface class hierarchy.

The Interface class hierarchy provides the part of a Model that is responsible for mapping a set of Variables into a set of Responses. The mapping is performed using either a simulation-based application interface or a surrogate-based approximation interface. For memory efficiency and enhanced polymorphism, the interface hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Interface) serves as the envelope and one of the derived classes (selected in Interface::get_interface()) serves as the letter.
13.68. INTERFACE CLASS REFERENCE

13.68.2 Constructor & Destructor Documentation

Interface ( )

default constructor
   used in Model envelope class instantiations

Interface ( ProblemDescDB & problem_db )

standard constructor for envelope
   Used in Model instantiation to build the envelope. This constructor only needs to extract enough data to properly execute get_interface, since Interface::Interface(BaseConstructor, problem_db) builds the actual base class data inherited by the derived interfaces.
   References Dakota::abort_handler(), Interface::get_interface(), and Interface::interfaceRep.

Interface ( const Interface & interface_in )

copy constructor
   Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.
   References Interface::interfaceRep, and Interface::referenceCount.

~Interface ( ) [virtual]

destructor
   Destructor decrements referenceCount and only deletes interfaceRep if referenceCount is zero.
   References Interface::interfaceRep, and Interface::referenceCount.

Interface ( BaseConstructor, const ProblemDescDB & problem_db ) [protected]

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
   This constructor is the one which must build the base class data for all inherited interfaces. get_interface() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_interface() again). Since this is the letter and the letter IS the representation, interfaceRep is set to NULL (an uninitialized pointer causes problems in ~Interface).
   References Dakota::abort_handler(), Interface::algebraic_function_type(), Interface::algebraicConstraintWeights, Interface::algebraicFnTags, Interface::algebraicFnTypes, Interface::algebraicMappings, Interface::algebraicVarTags, Interface::asl, ProblemDescDB::get_string(), Interface::outputLevel, and Dakota::strends().

13.68.3 Member Function Documentation

Interface operator= ( const Interface & interface_in )

assignment operator
   References Interface::interfaceRep, and Interface::referenceCount.
void assign_rep ( Interface * interface_rep, bool ref_count_incr = true )

assign letter or replace existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old interfaceRep and assigns the new interfaceRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.

- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_interface(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), Interface::interfaceRep, and Interface::referenceCount. Referenced by DataFitSurrModel::DataFitSurrModel(), parallel_interface_plugin(), LibraryEnvironment::plugin_interface(), and run_dakota().

void eval_tag_prefix ( const String & eval_id_str, bool append_iface_id = true )

set the evaluation tag prefix (does not recurse)

default implementation just sets the list of eval ID tags; derived classes containing additional models or interfaces should override (currently no use cases)

References Interface::appendIfaceId, Interface::eval_tag_prefix(), Interface::evalTagPrefix, and Interface::interfaceRep.

Referenced by NestedModel::derived_evaluate(), SimulationModel::eval_tag_prefix(), and Interface::eval_tag_prefix().

void response_mapping ( const Response & algebraic_response, const Response & core_response, Response & total_response ) [protected]

combine the response from algebraic_mappings() with the response from derived_map() to create the total response

This function will get invoked even when only algebraic mappings are active (no core mappings from derived_map), since the AMPL algebraic_response may be ordered differently from the total_response. In this case, the core_response object is unused.

References Dakota::NPOS, Dakota::abort_handler(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), Interface::algebraicACVIds, Interface::algebraicFnIndices, Interface::coreMappings, Dakota::find_index(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessian_view(), Response::function_hessians(), Response::function_value(), Response::function_values(), Response::function_values_view(), Interface::outputLevel, Response::reset(), and Response::reset_inactive().

Referenced by ApproximationInterface::map(), ApplicationInterface::map(), ApplicationInterface::synchronize(), and ApplicationInterface::synchronize_nowait().

Interface * get_interface ( ProblemDescDB & problem_db ) [private]

Used by the envelope to instantiate the correct letter class.

used only by the envelope constructor to initialize interfaceRep to the appropriate derived type.
13.69. **ITERATOR CLASS REFERENCE**

References ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), and Interface::interface_type().
Referenced by Interface::Interface().

13.68.4 **Member Data Documentation**

**IntResponseMap rawResponseMap** [protected]

Set of responses returned by either a blocking or nonblocking schedule.

The map is a full/partial set of completions which are identified through their evalIdCntr key. The raw set is postprocessed (i.e., finite diff grads merged) in Model::synchronize() where it becomes responseMap.

Referenced by ApplicationInterface::asynchronous_local_evaluations(), Interface::cache_unmatched_response(), ApplicationInterface::receive_evaluation(), ApplicationInterface::process_synch_local(), ApplicationInterface::synchronize_nowait(), ApplicationInterface::synchronize(), ApproximationInterface::synchronize(), ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

The documentation for this class was generated from the following files:

- DakotaInterface.hpp
- DakotaInterface.cpp

13.69 **Iterator Class Reference**

Base class for the iterator class hierarchy.

Inheritance diagram for Iterator:

```
   +-----------------------------------------+  +-------------------------+
   | Iterator                               |  | MetaIterator             |
   '----------------------------------------'  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | Minimizer                |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | LeastSq                  |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | Optimizer                |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | SurBaseMinimizer         |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | SeqHybridMetaIterator    |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | CollabHybridMetaIterator |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | EmbedHybridMetaIterator  |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | NonD                     |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | PlnhyDACE                 |
          |                                    |  '-------------------------'
          |                                    |  +-------------------------+
          |                                    |  | Verification             |
          |                                    |  '-------------------------'
```

Public Member Functions

- **Iterator** (std::shared_ptr<TraitsBase> traits=new TraitsBase())
  
  *default constructor*

- **Iterator** (std::shared_ptr<TraitsBase> traits=new TraitsBase())
  
  *alternate envelope constructor that assigns a representation pointer*

- **Iterator** (std::shared_ptr<TraitsBase> traits=new TraitsBase())
  
  *standard envelope constructor, which constructs its own model(s)*

- **Iterator** (std::shared_ptr<TraitsBase> traits=new TraitsBase())
  
  *alternate envelope constructor which uses the ProblemDescDB but accepts a model from a higher level (meta-iterator) context, instead of constructing its own*
• **Iterator** (const String &method_string, Model &model, std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  alternate envelope constructor for instantiations by name without the ProblemDescDB

• **Iterator** (const Iterator &iterator)
  copy constructor

• **virtual** ~*Iterator* ()
  destructor

• **Iterator** operator= (const Iterator &iterator)
  assignment operator

• **virtual** void derived_set_communicators (ParLevIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance

• **virtual** void derived_free_communicators (ParLevIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance

• **virtual** void initialize_run ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

• **virtual** void pre_run ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

• **virtual** void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

• **virtual** void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

• **virtual** void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

• **virtual** void pre_output ()
  write variables to file, following pre-run

• **virtual** void post_input ()
  read tabular data for post-run mode

• **virtual** void reset ()
  restore initial state for repeated sub-iterator executions

• **virtual** void initialize_iterator (int job_index)
  used by IteratorScheduler to set the starting data for a run

• **virtual** void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack starting data for an iterator run

• **virtual** void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack starting data for an iterator run

• **virtual** void unpack_parameters_initialize (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack starting data and initialize an iterator run

• **virtual** void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack results data from an iterator run

• **virtual** void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack results data from an iterator run
• virtual void update_local_results (int job_index)
  used by IteratorScheduler to update local results arrays

• virtual const Variables & variables_results () const
  return a single final iterator solution (variables)

• virtual const Response & response_results () const
  return a single final iterator solution (response)

• virtual const VariablesArray & variables_array_results ()
  return multiple final iterator solutions (variables). This should only be used if returns_multiple_points() returns true.

• virtual const ResponseArray & response_array_results ()
  return multiple final iterator solutions (response). This should only be used if returns_multiple_points() returns true.

• virtual void response_results_active_set (const ActiveSet &set)
  set the requested data for the final iterator response results

• virtual const RealVector & response_error_estimates () const
  return error estimates associated with the final iterator solution

• virtual bool accepts_multiple_points () const
  indicates if this iterator accepts multiple initial points. Default return is false. Override to return true if appropriate.

• virtual bool returns_multiple_points () const
  indicates if this iterator returns multiple final points. Default return is false. Override to return true if appropriate.

• virtual void initial_points (const VariablesArray &pts)
  sets the multiple initial points for this iterator. This should only be used if accepts_multiple_points() returns true.

• virtual void initialize_graphics (int iterator_server_id=1)
  initialize the 2D graphics window and the tabular graphics data

• virtual void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results

• virtual const Model & algorithm_space_model () const
  return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain

• virtual unsigned short uses_method () const
  return name of any enabling iterator used by this iterator

• virtual void method_recourse ()
  perform a method switch, if possible, due to a detected conflict

• virtual const VariablesArray & all_variables ()
  return the complete set of evaluated variables

• virtual const RealMatrix & all_samples ()
  return the complete set of evaluated samples

• virtual const IntResponseMap & all_responses () const
  return the complete set of computed responses

• virtual int num_samples () const
  get the current number of samples

• virtual void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator to use at least min_samples

• virtual void sampling_reference (int samples_ref)
set reference number of samples, which is a lower bound during reset

- virtual void sampling_increment ()
  increment to next in sequence of refinement samples
- virtual unsigned short sampling_scheme () const
  return sampling name
- virtual bool compact_mode () const
  returns Analyzer::compactMode
- virtual IntIntPair estimate_partition_bounds ()
  estimate the minimum and maximum partition sizes that can be utilized by this Iterator
- virtual bool resize ()
  reinitializes iterator based on new variable size

- void init_communicators (ParLevLIter pl_iter)
  initialize the communicators associated with this Iterator instance
- void set_communicators (ParLevLIter pl_iter)
  set the communicators associated with this Iterator instance
- void free_communicators (ParLevLIter pl_iter)
  free the communicators associated with this Iterator instance
- void resize_communicators (ParLevLIter pl_iter, bool reinit_comms)
  Resize the communicators. This is called from the letter's resize()
- void parallel_configuration_iterator (ParConfigLIter pc_iter)
  set methodPCIter
- ParConfigLIter parallel_configuration_iterator () const
  return methodPCIter
- void run (ParLevLIter pl_iter)
  invoke set_communicators(pl_iter) prior to run()
- void run ()
  orchestrate initialize/pre/core/post/finalize phases
- void assign_rep (Iterator *iterator_rep, bool ref_count_incr=true)
  replaces existing letter with a new one
- void iterated_model (const Model &model)
  set the iteratedModel (iterators and meta-iterators using a single model instance)
- Model & iterated_model ()
  return the iteratedModel (iterators & meta-iterators using a single model instance)
- ProblemDescDB & problem_description_db () const
  return the problem description database (probDescDB)
- ParallelLibrary & parallel_library () const
  return the parallel library (parallelLib)
- void method_name (unsigned short m_name)
  set the method name to an enumeration value
- unsigned short method_name () const
  return the method name via its native enumeration value
- void method_string (const String &m_str)
  set the method name by string
• String method_string () const
  return the method name by string
• String method_enum_to_string (unsigned short method_name) const
  convert a method name enumeration value to a string
• unsigned short method_string_to_enum (const String &method_name) const
  convert a method name string to an enumeration value
• String submethod_enum_to_string (unsigned short submethod_name) const
  convert a method name enumeration value to a string
• const String & method_id () const
  return the method identifier (methodId)
• int maximum_evaluation_concurrency () const
  return the maximum evaluation concurrency supported by the iterator
• void maximum_evaluation_concurrency (int max_conc)
  set the maximum evaluation concurrency supported by the iterator
• int maximum_iterations () const
  return the maximum iterations for this iterator
• void maximum_iterations (int max_iter)
  set the maximum iterations for this iterator
• void convergence_tolerance (Real conv_tol)
  set the method convergence tolerance (convergenceTol)
• Real convergence_tolerance () const
  return the method convergence tolerance (convergenceTol)
• void output_level (short out_lev)
  set the method output level (outputLevel)
• short output_level () const
  return the method output level (outputLevel)
• void summary_output (bool summary_output_flag)
  Set summary output control; true enables evaluation/results summary.
• size_t num_final_solutions () const
  return the number of solutions to retain in best variables/response arrays
• void num_final_solutions (size_t num_final)
  set the number of solutions to retain in best variables/response arrays
• void active_set (const ActiveSet &set)
  set the default active set vector (for use with iterators that employ evaluate_parameter_sets())
• const ActiveSet & active_set () const
  return the default active set vector (used by iterators that employ evaluate_parameter_sets())
• void sub_iterator_flag (bool si_flag)
  set subIteratorFlag (and update summaryOutputFlag if needed)
• void active_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
  set primaryA{CV,DIV,DRV}MapIndices, secondaryA{CV,DIV,DRV}MapTargets
• bool is_null () const
function to check iteratorRep (does this envelope contain a letter?)

- Iterator * iterator_rep () const
  returns iteratorRep for access to derived class member functions that are not mapped to the top Iterator level

- virtual void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix

- std::shared_ptr< TraitsBase > traits () const
  returns methodTraits for access to derived class member functions that are not mapped to the top TraitsBase level

Protected Member Functions

- Iterator (BaseConstructor, ProblemDescDB &problem_db, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >(new TraitsBase()))
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- Iterator (NoDBBaseConstructor, unsigned short method_name, Model &model, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >(new TraitsBase()))
  alternate constructor for base iterator classes constructed on the fly

- Iterator (NoDBBaseConstructor, unsigned short method_name, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >(new TraitsBase()))
  alternate constructor for base iterator classes constructed on the fly

- virtual void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance

- virtual void update_from_model (const Model &model)
  set inherited data attributes based on extractions from incoming model

- virtual const VariablesArray & initial_points () const
  gets the multiple initial points for this iterator. This will only be meaningful after a call to initial_points mutator.

- StrStrSizet run_identifier () const
  get the unique run identifier based on method name, id, and number of executions

Static Protected Member Functions

- static void gnewton_set_recast (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  conversion of request vector values for the Gauss-Newton Hessian approximation

Protected Attributes

- ProblemDescDB & probDescDB
  class member reference to the problem description database

- ParallelLibrary & parallelLib
  class member reference to the parallel library

- ParConfigLIter methodPCIter
  the active ParallelConfiguration used by this Iterator instance

- Model iteratedModel
  the model to be iterated (for iterators and meta-iterators employing a single model instance)
• size_t myModelLayers
  number of Models locally (in Iterator or derived classes) wrapped around the initially passed in Model

• unsigned short methodName
  name of the iterator (the user's method spec)

• Real convergenceTol
  iteration convergence tolerance

• int maxIterations
  maximum number of iterations for the iterator

• int maxFunctionEvals
  maximum number of fn evaluations for the iterator

• int maxEvalConcurrency
  maximum number of concurrent model evaluations

• ActiveSet activeSet
  the response data requirements on each function evaluation

• size_t numFinalSolutions
  number of solutions to retain in best variables/response arrays

• VariablesArray bestVariablesArray
  collection of N best solution variables found during the study; always in context of Model originally passed to the
  Iterator (any in-flight Recasts must be undone)

• ResponseArray bestResponseArray
  collection of N best solution responses found during the study; always in context of Model originally passed to the
  Iterator (any in-flight Recasts must be undone)

• bool subIteratorFlag
  flag indicating if this Iterator is a sub-iterator (NestedModel::subIterator or DataFitSurrModel::daceIterator)

• SizetArray primaryACVarMapIndices
  "primary" all continuous variable mapping indices flowed down from higher level iteration

• SizetArray primaryADIVarMapIndices
  "primary" all discrete int variable mapping indices flowed down from higher level iteration

• SizetArray primaryADSVarMapIndices
  "primary" all discrete string variable mapping indices flowed down from higher level iteration

• SizetArray primaryADRVarMapIndices
  "primary" all discrete real variable mapping indices flowed down from higher level iteration

• ShortArray secondaryACVarMapTargets
  "secondary" all continuous variable mapping targets flowed down from higher level iteration

• ShortArray secondaryADIVarMapTargets
  "secondary" all discrete int variable mapping targets flowed down from higher level iteration

• ShortArray secondaryADSVarMapTargets
  "secondary" all discrete string variable mapping targets flowed down from higher level iteration

• ShortArray secondaryADRVarMapTargets
  "secondary" all discrete real variable mapping targets flowed down from higher level iteration

• short outputLevel
  output verbosity level: {SILENT,QUIET,NORMAL,VERBOSE,DEBUG},OUTPUT

• bool summaryOutputFlag
flag for summary output (evaluation stats, final results); default true, but false for on-the-fly (helper) iterators and sub-iterator use cases

- **ResultsManager & resultsDB**
  reference to the global iterator results database

- **ResultsNames resultsNames**
  valid names for iterator results

- **std::shared_ptr< TraitsBase > methodTraits**
  pointer that retains shared ownership of a TraitsBase object, or child thereof

### Private Member Functions

- **Iterator * get_iterator (ProblemDescDB &problem_db)**
  Used by the envelope to instantiate the correct letter class.

- **Iterator * get_iterator (ProblemDescDB &problem_db, Model &model)**
  Used by the envelope to instantiate the correct letter class.

- **Iterator * get_iterator (const String &method_string, Model &model)**
  Used by the envelope to instantiate the correct letter class.

### Private Attributes

- **String methodId**
  method identifier string from the input file

- **size_t execNum**
  an execution number for this instance of the class, unique across all instances of same methodName/methodId

- **std::map< size_t, ParConfigLIter > methodPCIterMap**
  track the available configurations that have been created (init,communicators) and are available for activation at run time (set,communicators)

- **Iterator * iteratorRep**
  pointer to the letter (initialized only for the envelope)

- **int referenceCount**
  number of objects sharing iteratorRep

### 13.69.1 Detailed Description

Base class for the iterator class hierarchy.

The **Iterator** class is the base class for one of the primary class hierarchies in DAKOTA. The iterator hierarchy contains all of the iterative algorithms which use repeated execution of simulations as function evaluations. For memory efficiency and enhanced polymorphism, the iterator hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (**Iterator**) serves as the envelope and one of the derived classes (selected in **Iterator::get_iterator()**) serves as the letter.
### 13.69.2 Constructor & Destructor Documentation

**Iterator (std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()))**

default constructor

The default constructor is used in Vector<Iterator> instantiations and for initialization of Iterator objects contained in meta-Iterators and Model recursions. iteratorRep is NULL in this case, making it necessary to check for NULL pointers in the copy constructor, assignment operator, and destructor.

**Iterator (Iterator∗ iteratorRep, bool ref_count_incr = true, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()))**

alternate envelope constructor that assigns a representation pointer

This constructor assigns a representation pointer and optionally increments its reference count. It behaves the same as a default construction followed by assign_rep().

References Iterator::iteratorRep, and Iterator::referenceCount.

**Iterator (ProblemDescDB & problem_db, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()))**

standard envelope constructor, which constructs its own model(s)

Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for top-level ProblemDescDB-driven construction of all Iterators and MetaIterators, which construct their own Model instances.

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

**Iterator (ProblemDescDB & problem_db, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()))**

alternate envelope constructor which uses the ProblemDescDB but accepts a model from a higher level (meta-iterator) context, instead of constructing its own

Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for ProblemDescDB-driven construction of Iterators that are passed a Model from a higher-level context (e.g., a MetaIterator instantiates its sub-iterator(s) by name instead of pointer and passes in its iteratedModel, since these sub-iterators lack their own model pointers).

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

**Iterator (const String & method_string, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()))**

alternate envelope constructor for instantiations by name without the ProblemDescDB

Used in sub-iterator instantiations within iterator constructors. Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for lightweight constructions without the ProblemDescDB.

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

**Iterator (const Iterator & iterator)**

copy constructor

Copy constructor manages sharing of iteratorRep and incrementing of referenceCount.

References Iterator::iteratorRep, and Iterator::referenceCount.
~Iterator( ) [virtual]

destructor

Destructor decrements referenceCount and only deletes iteratorRep when referenceCount reaches zero.
References Iterator::iteratorRep, and Iterator::referenceCount.

Iterator ( BaseConstructor, ProblemDescDB & problem_db, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) ) [protected]

contstructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor builds the base class data for all inherited iterators, including meta-iterators. get_iterator() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_iterator() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Iterator).

References Iterator::method_enum_to_string(), Iterator::methodName, and Iterator::outputLevel.

Iterator ( NoDBBaseConstructor, unsigned short method_name, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) ) [protected]

alternate constructor for base iterator classes constructed on the fly

This alternate constructor builds base class data for inherited iterators. It is used for on-the-fly instantiations for which DB queries cannot be used, and is not used for construction of meta-iterators.

Iterator ( NoDBBaseConstructor, unsigned short method_name, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) ) [protected]

alternate constructor for base iterator classes constructed on the fly

This alternate constructor builds base class data for inherited iterators. It is used for on-the-fly instantiations for which DB queries cannot be used, and is not used for construction of meta-iterators. It has no incoming model, so only sets up a minimal set of defaults. However, its use is preferable to the default constructor, which should remain as minimal as possible.

13.69.3 Member Function Documentation

Iterator operator=( const Iterator & iterator )

assignment operator


References Iterator::iteratorRep, and Iterator::referenceCount.

void initialize_run( ) [virtual]

utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.

Reimplemented in Optimizer, NonD, Analyzer, Minimizer, and LeastSq.

References Iterator::initialize_run(), and Iterator::iteratorRep.

Referenced by Iterator::initialize_run(), Iterator::run(), and SeqHybridMetaIterator::run_sequential_adaptive().
void pre_run() [virtual]
pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
  pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
  Reimplemented in NonDSampling, NonD, Analyzer, NonDLHSSampling, DDACEDesignCompExp, ConcurrentMetaIterator, NonDRKDDarts, FSUDesignCompExp, ParamStudy, NonDMultilevelSampling, PSUADEDesignCompExp, and SurrBasedLocalMinimizer.
  References Iterator::iteratorRep, and Iterator::pre_run().
  Referenced by NonDBayesCalibration::calibrate_to_hifi(), Iterator::pre_run(), and Iterator::run().

void core_run() [virtual]
core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post steps
  Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
  References Dakota::abort_handler(), Iterator::core_run(), and Iterator::iteratorRep.
  Referenced by Iterator::core_run(), and Iterator::run().

void post_run(std::ostream & s) [virtual]
post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
  Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.
  Reimplemented in Optimizer, NonDRKDDarts, Analyzer, Minimizer, NonDLHSSampling, DDACEDesignCompExp, FSUDesignCompExp, MetaIterator, ParamStudy, PSUADEDesignCompExp, SurrBasedLocalMinimizer, NonDMultilevelSampling, and LeastSq.
  References Iterator::iteratorRep, and Iterator::post_run().
  Referenced by Iterator::post_run(), and Iterator::run().

void finalize_run() [virtual]
utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
  Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.
  Reimplemented in Optimizer, NonD, Analyzer, Minimizer, and LeastSq.
  References Iterator::finalize_run(), and Iterator::iteratorRep.
  Referenced by Iterator::finalize_run(), Minimizer::finalize_run(), Analyzer::finalize_run(), Iterator::run(), and SeqHybridMetaIterator::run_sequential_adaptive().
void initialize_graphics ( int iterator_server_id = 1 ) [virtual]

initialize the 2D graphics window and the tabular graphics data

This is a convenience function for encapsulating graphics initialization operations. It is overridden by derived classes that specialize the graphics display.

Reimplemented in NonDReliability, and SurrBasedMinimizer.

References Model::auto_graphics(), Graphics::create_plots_2d(), OutputManager::create_tabular_datastream(), Model::current_response(), Model::current_variables(), OutputManager::graph2DFlag, OutputManager::graphics(), Iterator::initialize_graphics(), Iterator::iteratedModel, Iterator::iteratorRep, ParallelLibrary::output_manager(), Iterator::parallelLib, and OutputManager::tabularDataFlag.

Referred by CollabHybridMetaIterator::core_run(), EmbedHybridMetaIterator::core_run(), ConcurrentMetaIterator::core_run(), Environment::execute(), Iterator::initialize_graphics(), SeqHybridMetaIterator::run_sequential(), and SeqHybridMetaIterator::run_sequential_adaptive().

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented in Optimizer, NonDPolynomialChaos, Analyzer, NonDGPMSABayesCalibration, NonDLHSSampling, NonDBayesCalibration, NonDPODarts, NonDMultilevelPolynomialChaos, NonDLocalReliability, NonDWASABIBayesCalibration, NonDQUESTOBayesCalibration, SeqHybridMetaIterator, NonDAdaptiveSampling, ConcurrentMetaIterator, NonDGPImpSampling, NonDAdaptImpSampling, NonDExpansion, NonDInterval, NonDMultilevelSampling, PStudyDACE, LeastSq, Verification, NonDGlobalReliability, SurrBasedMinimizer, and RichExtrapVerification.

References Iterator::iteratorRep, and Iterator::print_results().

Referred by MetaIterator::post_run(), Minimizer::post_run(), and Iterator::print_results().

unsigned short uses_method ( ) const [virtual]

return name of any enabling iterator used by this iterator

This is used to avoid clashes in state between non-object-oriented (i.e., F77, C) iterator executions, when such iterators could potentially be executing simultaneously (e.g., nested execution). It is not an issue (and a used method is not reported) in cases where a helper execution is completed before a lower level one could be initiated; an example of this is DIRECT for maximization of expected improvement: the EIF maximization is completed before a new point evaluation (which could include nested iteration) is performed.

Reimplemented in NonDLocalReliability, and NonDLocalInterval.

References Iterator::iteratorRep, and Iterator::uses_method().

Referred by NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), S-OLBase::SOLBase(), and Iterator::uses_method().

void run ( )

orchestrate initialize/pre/core/post/finalize phases

Iterator supports a construct/initialize-run/pre-run/core-run/post-run/finalize-run/destroy progression. This member (non-virtual) function sequences these run phases.

References ParallelLibrary::command_line_post_run(), ParallelLibrary::command_line_pre_run(), ParallelLibrary::command_line_run(), Iterator::core_run(), Iterator::execNum, Iterator::finalize_run(), ResultsID::increment_id(),
void assign_rep ( Iterator * iterator_rep, bool ref_count_incr = true )

replaces existing letter with a new one.

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old iterator-Rep and assigns the new iteratorRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.
- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_iterator(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), Iterator::iterator_rep(), and Iterator::referenceCount. Referenced by NonDBayesCalibration::calibrate_to_hifi(), NonDExpansion::construct_cubature(), NonDExpansion::construct_expansion_sampler(), NonDAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), NonDBayesCalibration::construct_map_optimizer(), NonDBayesCalibration::construct_mcmc_model(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), Minimizer::data.transform_model(), EffGlobalMinimizer::EffGlobalMinimizer(), ActiveSubspaceModel::init_fullspace_samples(), NonDLocalInterval::method_recourse(), NonDLocalReliability::method_recourse(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGPMsABayesCalibration::NonDGPMsABayesCalibration(), NonDInterval::NonDInterval(), NonDLocalReliability::NonDLocalReliability(), GaussProcApproximation::optimize_theta_global(), GaussProcApproximation::optimize_theta_multipoint(), and SurrBasedLocalMinimizer::relax_constraints().
References ActiveSet::request_value(), and ActiveSet::request_vector().
Referenced by NonDBayesCalibration::construct_map_optimizer(), and Optimizer::reduce_model().

**Iterator * get_iterator ( ProblemDescDB & problem_db ) [private]**

Used by the envelope to instantiate the correct letter class.

Used only by the envelope constructor to initialize iteratorRep to the appropriate derived type, as given by the DB’s method_name. Supports all iterators and meta-iterators. These instantiations will NOT recurse on the Iterator(problem_db) constructor due to the use of BaseConstructor.

References ProblemDescDB::get_model(), ProblemDescDB::get_ushort(), Iterator::method_enum_to_string(), Iterator::method_name(), and Dakota::SUBMETHOD_COLLABORATIVE.

Referenced by Iterator::Iterator().

**Iterator * get_iterator ( ProblemDescDB & problem_db, Model & model ) [private]**

Used by the envelope to instantiate the correct letter class.

Used only by the envelope constructor to initialize iteratorRep to the appropriate derived type. Alternate construction of meta-iterators is supported to enable use of meta-iterators as components. These instantiations will NOT recurse on the Iterator(problem_db, model) constructor due to the use of BaseConstructor.

References ProblemDescDB::get_ushort(), Iterator::method_enum_to_string(), Iterator::method_name(), Iterator::probDescDB, Dakota::SUBMETHOD_COLLABORATIVE, and Model::surrogate_type().

**Iterator * get_iterator ( const String & method_string, Model & model ) [private]**

Used by the envelope to instantiate the correct letter class.

Used only by the envelope constructor to initialize iteratorRep to the appropriate derived type, as given by the passed method_string. Lightweight instantiations by name are supported by a subset of Iterators (primarily Minimizers).

References Dakota::strbegins(), and Dakota::strends().

**13.69.4 Member Data Documentation**

**ProblemDescDB& probDescDB [protected]**

class member reference to the problem description database

**Iterator** and **Model** cannot use a shallow copy of ProblemDescDB due to circular destruction dependency (reference counts can’t get to 0), since ProblemDescDB contains {iterator,model} List.

Referenced by MetaIterator::allocate_by_name(), MetaIterator::allocate_by_pointer(), Analyzer::Analyzer(), MetaIterator::check_model(), NonDBayesCalibration::construct_mcmc_model(), Minimizer::data_transform_model(), SurrBasedMinimizer::derived_init_communicators(), EmbedHybridMetaIterator::derived_init_communicators(), ConcurrentMetaIterator::derived_init_communicators(), EffGlobalMinimizer::EffGlobalMinimizer(), MetaIterator::estimate_by_name(), MetaIterator::estimate_by_pointer(), CollabHybridMetaIterator::estimate_partition_bounds(), EmbedHybridMetaIterator::estimate_partition_bounds(), ConcurrentMetaIterator::estimate_partition_bounds(), SeqHybridMetaIterator::estimate_partition_bounds(), FSUDesignCompExp::FSUDesignCompExp(), Iterator::get_iterator(), ConcurrentMetaIterator::initialize_model(), NOWPACOptimizer::initialize_options(), SurrBasedLocalMinimizer::initialize_sub_minimizer(), JEGAOptimizer::JEGAOptimizer(), NonD::NonD(), NonDAdaptImpSampling::NonDAdaptImpSampling(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGpmsABayesCalibration::NonDGpmsABayesCalibration(), NonDImpSampling::NonDImpSampling(), NonDInterval::NonDInterval(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDGpmsABayesCalibration::NonDGpmsABayesCalibration(), NonDAdaptImpSampling::NonDAdaptImpSampling(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGPlmpSampling::NonDGPlmpSampling(), NonDGpmsABayesCalibration::NonDGpmsABayesCalibration(), NonDImpSampling::NonDImpSampling(), NonDInterval::NonDInterval(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability().
13.70. **ITERATORSCHEDULER CLASS REFERENCE**

This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g., meta-iteration, nested models).

### Public Member Functions

- **IteratorScheduler** (`ParallelLibrary` &`parallel_lib`, bool `peer_assign_jobs`, int `num_servers`=0, int `procs_per_iterator`=0, short `scheduling`=DEFAULT_SCHEDULING)
CHAPTER 13. CLASS DOCUMENTATION

• constructor
• destructor
• void construct_sub_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model, const String &method_ptr, const String &method_name, const String &model_ptr)
  instantiate sub_iterator on the current rank if not already constructed
• IntIntPair configure (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model)
  performs sufficient initialization to define partitioning controls (min and max processors per iterator server)
• IntIntPair configure (ProblemDescDB &problem_db, const String &method_string, Iterator &sub_iterator, Model &sub_model)
  performs sufficient initialization to define partitioning controls (min and max processors per iterator server)
• void partition (int max_iterator_concurrency, IntIntPair &ppi_pr)
  convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart.
• void init_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model)
  invokes static version of this function with appropriate parallelism level
• void init_iterator (ProblemDescDB &problem_db, const String &method_string, Iterator &sub_iterator, Model &sub_model)
  invokes static version of this function with appropriate parallelism level
• void set_iterator (Iterator &sub_iterator)
  invokes static version of this function with appropriate parallelism level
• void run_iterator (Iterator &sub_iterator)
  invokes static version of this function with appropriate parallelism level
• void free_iterator (Iterator &sub_iterator)
  invokes static version of this function with appropriate parallelism level
• void free_iterator_parallelism ()
  convenience function for deallocating the concurrent iterator parallelism level
• template<typename MetaType >
  void schedule_iterators (MetaType &meta_object, Iterator &sub_iterator)
  short convenience function for distributing control among master_dynamic_schedule_iterators(), serve_iterators(), and peer_static_schedule_iterators()
• template<typename MetaType >
  void master_dynamic_schedule_iterators (MetaType &meta_object)
  executed by the scheduler master to manage a dynamic schedule of iterator jobs among slave iterator servers
• void stop_iterator_servers ()
  executed by the scheduler master to terminate slave iterator servers
• template<typename MetaType >
  void serve_iterators (MetaType &meta_object, Iterator &sub_iterator)
  executed on the slave iterator servers to perform iterator jobs assigned by the scheduler master
• template<typename MetaType >
  void peer_static_schedule_iterators (MetaType &meta_object, Iterator &sub_iterator)
  executed on iterator peers to manage a static schedule of iterator jobs
13.70. ITERATORSCHEDULER CLASS REFERENCE

- void update (ParConfigLIter pc_iter)
  update schedPCIter
- void update (size_t index)
  update mPlIndex as well as associated settings for concurrent iterator scheduling from the corresponding Parallel-Level
- void update (ParConfigLIter pc_iter, size_t index)
  invoke update(ParConfigLIter) and update(size_t) in sequence
- void iterator_message_lengths (int params_msg_len, int results_msg_len)
  update paramsMsgLen and resultsMsgLen
- bool lead_rank () const
  determines if current processor is rank 0 of the parent comm

Static Public Member Functions

- static void init_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, ParLevLIter pl_iter)
  convenience function for allocation of an iterator and (parallel) initialization of its comms
- static void init_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model, ParLevLIter pl_iter)
  convenience function for allocation of an iterator and (parallel) initialization of its comms
- static void init_iterator (ProblemDescDB &problem_db, const String &method_string, Iterator &sub_iterator, Model &sub_model, ParLevLIter pl_iter)
  convenience function for lightweight allocation of an iterator and (parallel) initialization of its comms
- static void set_iterator (Iterator &sub_iterator, ParLevLIter pl_iter)
  convenience function for setting comms prior to running an iterator
- static void run_iterator (Iterator &sub_iterator, ParLevLIter pl_iter)
  Convenience function for invoking an iterator and managing parallelism. This version omits communicator reparation. Function must be public due to use by MINLPNode.
- static void free_iterator (Iterator &sub_iterator, ParLevLIter pl_iter)
  convenience function for deallocating comms after running an iterator

Public Attributes

- ParallelLibrary & parallelLib
  reference to the ParallelLibrary instance
- int numIteratorJobs
  number of iterator executions to schedule
- int numIteratorServers
  number of concurrent iterator partitions
- int procsPerIterator
  partition size request
- int iteratorCommRank
  processor rank in iteratorComm
- int iteratorCommSize
  number of processors in iteratorComm
- int iteratorServerId
CHAPTER 13. CLASS DOCUMENTATION

412

identifier for an iterator server
• bool messagePass
  flag for message passing among iterator servers
• short iteratorScheduling
  \{DEFAULT,MASTER,PEER\}_SCHEDULING
• bool peerAssignJobs
  flag indicating need for peer 1 to assign jobs
  < to peers 2-n
• ParConfigLIter schedPCIter
  iterator for active parallel configuration
• size_t miPLIndex
  index of active parallel level (corresponding
  < to ParallelConfiguration::miPLIters) to use < for parallelLib send/recv

Private Attributes
• int paramsMsgLen
  length of MPI buffer for parameter input instance(s)
• int resultsMsgLen
  length of MPI buffer for results output instance(s)

13.70.1 Detailed Description
This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g.,
meta-iteration, nested models).

In time, a Scheduler class hierarchy is envisioned, but for now, this class is not part of a hierarchy.

13.70.2 Constructor & Destructor Documentation
IteratorScheduler ( ParallelLibrary & parallelLib, bool peerAssignJobs, int numServers = 0, int
procsPerIterator = 0, short scheduling = DEFAULT_SCHEDULING )
constructor
  Current constructor parameters are the input specification components, which are requests subject to override
by ParallelLibrary::init_iterator_communicators().

13.70.3 Member Function Documentation
void init_iterator ( ProblemDescDB & problemDb, Iterator & subIterator, ParLevLIter plIter )
  [static]
convenience function for allocation of an iterator and (parallel) initialization of its comms
  This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.
  References ProblemDescDB::get_iterator(), ProblemDescDB::get_model(), ProblemDescDB::get_ushort(), Model-
::init_comms_bcast_flag(), Iterator::init_communicators(), Iterator::is_null(), Model::is_null(), Iterator::iterated_-model(), Iterator::maximum_evaluation_concurrency(), Iterator::method_name(), Model::serve_init_communicators(), and
Model::stop_init_communicators().
  Referenced by MetaIterator::allocate_by_name(), MetaIterator::allocate_by_pointer(), Environment::construct(),
ConcurrentMetaIterator::derived_init_communicators(), NestedModel::derived_init_communicators(), and Iterator-
Scheduler::init_iterator().
void init_iterator (ProblemDescDB & problem_db, Iterator & sub_iterator, Model & sub_model, ParLevLIter pl_iter) [static]

convenience function for allocation of an iterator and (parallel) initialization of its comms

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.

References ProblemDescDB::get_iterator(), ProblemDescDB::get_ushort(), Model::init_comms_bcast_flag(), Iterator::init_communicators(), Iterator::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluationConcurrency(), Iterator::method_name(), Model::serve_init_communicators(), and Model::stop_init_communicators().

void init_iterator (ProblemDescDB & problem_db, const String & method_string, Iterator & sub_iterator, Model & sub_model, ParLevLIter pl_iter) [static]

convenience function for lightweight allocation of an iterator and (parallel) initialization of its comms

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.

References ProblemDescDB::get_iterator(), Model::init_comms_bcast_flag(), Iterator::init_communicators(), Iterator::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluationConcurrency(), Iterator::method_string(), Model::serve_init_communicators(), and Model::stop_init_communicators().

void set_iterator (Iterator & sub_iterator, ParLevLIter pl_iter) [static]

convenience function for setting comms prior to running an iterator

This is a convenience function for encapsulating the deallocation of communicators after running an iterator.

References Iterator::derived_set_communicators(), and Iterator::set_communicators().

Referenced by CollabHybridMetaIterator::derived_set_communicators(), EmbedHybridMetaIterator::derived_set_communicators(), ConcurrentMetaIterator::derived_set_communicators(), SeqHybridMetaIterator::derived_set_communicators(), NestedModel::derived_set_communicators(), and IteratorScheduler::set_iterator().

void run_iterator (Iterator & sub_iterator, ParLevLIter pl_iter) [static]

Convenience function for invoking an iterator and managing parallelism. This version omits communicator repartitioning. Function must be public due to use by MINLPNode.

This is a convenience function for encapsulating the parallel features (run/serve) of running an iterator. This function omits allocation/deallocation of communicators to provide greater efficiency in approaches that involve multiple iterator executions but only require communicator allocation/deallocation to be performed once.

References Model::finalize_mapping(), Model::initialize_mapping(), Iterator::iterated_model(), Iterator::maximum_evaluationConcurrency(), Iterator::method_name(), Iterator::resize(), Iterator::resize_communicators(), Iterator::run(), Model::serve_finalize_mapping(), Model::serve_init_mapping(), Model::serve_run(), Model::stop_finalize_mapping(), Model::stop_init_mapping(), and Model::stop_servers().

Referenced by NestedModel::derived_evaluate(), Environment::execute(), IteratorScheduler::peer_static_schedule_iterators(), IteratorScheduler::run_iterator(), and IteratorScheduler::serve_iterators().

void free_iterator (Iterator & sub_iterator, ParLevLIter pl_iter) [static]

convenience function for deallocating comms after running an iterator

This is a convenience function for encapsulating the deallocation of communicators after running an iterator.

References Iterator::derived_free_communicators(), Iterator::free_communicators(), and Iterator::method_name().

Referenced by CollabHybridMetaIterator::derived_free_communicators(), EmbedHybridMetaIterator::derived_free_communicators(), ConcurrentMetaIterator::derived_free_communicators(), SeqHybridMetaIterator::derived_free_communicators(), NestedModel::derived_free_communicators(), Environment::destruct(), and IteratorScheduler::free_iterator().
IntIntPair configure ( ProblemDescDB & problem_db, Iterator & sub_iterator, Model & sub_model )

performs sufficient initialization to define partitioning controls (min and max processors per iterator server).
This is a convenience function for computing the minimum and maximum partition size prior to concurrent iterator partitioning.
References ProblemDescDB::get_iterator(), IteratorScheduler::schedPCIter, and ParallelLevel::server_communicator_rank(). Referenced by IteratorScheduler::configure(), ConcurrentMetaIterator::derived_init_communicators(), NestedModel::derived_init_communicators(), MetaIterator::estimate_by_name(), and MetaIterator::estimate_by_pointer().

IntIntPair configure ( ProblemDescDB & problem_db, const String & method_string, Iterator & sub_iterator, Model & sub_model )

performs sufficient initialization to define partitioning controls (min and max processors per iterator server).
This is a convenience function for computing the minimum and maximum partition size prior to concurrent iterator partitioning.
References IteratorScheduler::configure(), ProblemDescDB::get_iterator(), IteratorScheduler::schedPCIter, and ParallelLevel::server_communicator_rank().

IntIntPair configure ( ProblemDescDB & problem_db, Iterator & sub_iterator )

performs sufficient initialization to define partitioning controls (min and max processors per iterator server).
This is a convenience function for computing the minimum and maximum partition size prior to concurrent iterator partitioning.
References ParallelLibrary::bcast(), Iterator::estimate_partition_bounds(), ProblemDescDB::get_db_method_node(), ProblemDescDB::get_db_model_node(), IteratorScheduler::parallelLib, IteratorScheduler::schedPCIter, ParallelLevel::server_communicator_rank(), ParallelLevel::server_communicator_size(), ProblemDescDB::set_db_method_node(), ProblemDescDB::set_db_model_nodes(), and MPIPackBuffer::size().

void partition ( int max_iterator_concurrency, IntIntPair & ppi_pr )

convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart.
Called from derived class constructors once maxIteratorConcurrency is defined but prior to instantiating Iterators and Models.
References ParallelLibrary::init_iterator_communicators(), IteratorScheduler::iteratorScheduling, IteratorScheduler::numIteratorServers, ParallelLibrary::parallel_configuration_iterator(), IteratorScheduler::parallelLib, IteratorScheduler::procsPerIterator, ParallelLibrary::push_output_tag(), and IteratorScheduler::update(). Referenced by CollabHybridMetaIterator::derived_init_communicators(), EmbedHybridMetaIterator::derived_init_communicators(), ConcurrentMetaIterator::derived_init_communicators(), SeqHybridMetaIterator::derived_init_communicators(), and NestedModel::derived_init_communicators().

void schedule_iterators ( MetaType & meta_object, Iterator & sub_iterator )

short convenience function for distributing control among master_dynamic_schedule_iterators(), serve_iterators(), and peer_static_schedule_iterators().
This implementation supports the scheduling of multiple jobs using a single iterator/model pair. Additional future (overloaded) implementations could involve independent iterator instances.
References IteratorScheduler::iteratorScheduling, IteratorScheduler::iteratorServerId, IteratorScheduler::lead_rank(), IteratorScheduler::master_dynamic_schedule_iterators(), IteratorScheduler::numIteratorServers, ParallelLibrary::parallel_configuration_iterator(), IteratorScheduler::parallelLib, IteratorScheduler::peer_static_schedule_iterators(), IteratorScheduler::serve_iterators(), and IteratorScheduler::stop_iterator_servers().

Referenced by CollabHybridMetaIterator::core_run(), EmbedHybridMetaIterator::core_run(), ConcurrentMetaIterator::core_run(), NestedModel::derived_synchronize(), SeqHybridMetaIterator::run_sequential(), and NestedModel::serve_run().

```c++
void master_dynamic_schedule_iterators ( MetaType & meta_object )
```
executed by the scheduler master to manage a dynamic schedule of iterator jobs among slave iterator servers

This function is adapted from ApplicationInterface::master_dynamic_schedule_evaluations().

References ParallelLibrary::free(), ParallelLibrary::irecv_mi(), ParallelLibrary::isend_mi(), IteratorScheduler::miPLIndex, IteratorScheduler::numIteratorJobs, IteratorScheduler::numIteratorServers, ParallelLib::parallelLib, MPIPackBuffer::reset(), MPIUnpackBuffer::resize(), IteratorScheduler::resultsMsgLen, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by IteratorScheduler::schedule_iterators().

```c++
void serve_iterators ( MetaType & meta_object, Iterator & sub_iterator )
```
executed on the slave iterator servers to perform iterator jobs assigned by the scheduler master

This function is similar in structure to ApplicationInterface::serve_evaluations_synch().

References ParallelLibrary::bcast_i(), IteratorScheduler::iteratorCommRank, IteratorScheduler::iteratorCommSize, IteratorScheduler::miPLIndex, ParallelLibrary::parallel_time(), IteratorScheduler::parallelLib, IteratorScheduler::paramsMsgLen, ParallelLibrary::recv_mi(), IteratorScheduler::resultsMsgLen, IteratorScheduler::run_iterator(), and ParallelLibrary::send_mi().

Referenced by IteratorScheduler::schedule_iterators().

The documentation for this class was generated from the following files:

- IteratorScheduler.hpp
- IteratorScheduler.cpp

### 13.71 JEGAOptimizer Class Reference

A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

Inheritance diagram for JEGAOptimizer:
Classes

- class Driver
  A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.
- class Evaluator
  An evaluator specialization that knows how to interact with Dakota.
- class EvaluatorCreator
  A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

Public Member Functions

- virtual void core_run ()
  Performs the iterations to determine the optimal set of solutions.
- virtual bool accepts_multiple_points () const
  Overridden to return true since JEGA algorithms can accept multiple initial points.
- virtual bool returns_multiple_points () const
  Overridden to return true since JEGA algorithms can return multiple final points.
- virtual void initial_points (const VariablesArray &pts)
  Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables.
- virtual const VariablesArray & initial_points () const
  Overridden to return the collection of initial points for the JEGA algorithm created and run by this JEGAOptimizer.
- JEGAOptimizer (ProblemDescDB &problem_db, Model &model)
  Constructs a JEGAOptimizer class object.
- ~JEGAOptimizer ()
  Destructs a JEGAOptimizer.

Protected Member Functions

- void LoadDakotaResponses (const JEGA::Utilities::Design &from, Variables &vars, Response &resp) const
  Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects.
- void ReCreateTheParameterDatabase ()
  Destroys the current parameter database and creates a new empty one.
- void LoadTheParameterDatabase ()
  Reads information out of the known Dakota::ProblemDescDB and puts it into the current parameter database.
- void LoadAlgorithmConfig (JEGA::FrontEnd::AlgorithmConfig &aConfig)
  Completely initializes the supplied algorithm configuration.
- void LoadProblemConfig (JEGA::FrontEnd::ProblemConfig &pConfig)
  Completely initializes the supplied problem configuration.
- void LoadTheDesignVariables (JEGA::FrontEnd::ProblemConfig &pConfig)
  Adds DesignVariableInfo objects into the problem configuration object.
- void LoadTheObjectiveFunctions (JEGA::FrontEnd::ProblemConfig &pConfig)
  Adds ObjectiveFunctionInfo objects into the problem configuration object.
- void LoadTheConstraints (JEGA::FrontEnd::ProblemConfig &pConfig)
  Adds ConstraintInfo objects into the problem configuration object.
void GetBestSolutions (const JEGA::Utilities::DesignOFSortSet &from, const JEGA::Algorithms::GeneticAlgorithm &theGA, std::multimap<RealRealPair, JEGA::Utilities::Design*> &designSortMap)

Returns up to _numBest designs sorted by DAKOTA's fitness (L2 constraint violation, then utopia or objective), taking into account the algorithm type. The front of the returned map can be viewed as a single "best".

void GetBestMOSolutions (const JEGA::Utilities::DesignOFSortSet &from, const JEGA::Algorithms::GeneticAlgorithm &theGA, std::multimap<RealRealPair, JEGA::Utilities::Design*> &designSortMap)

Retrieve the best Designs from a set of solutions assuming that they are generated by a multi objective algorithm.

void GetBestSOSolutions (const JEGA::Utilities::DesignOFSortSet &from, const JEGA::Algorithms::GeneticAlgorithm &theGA, std::multimap<RealRealPair, JEGA::Utilities::Design*> &designSortMap)

Retrieve the best Designs from a set of solutions assuming that they are generated by a single objective algorithm.

JEGA::DoubleMatrix ToDoubleMatrix (const VariablesArray &variables) const

Converts the items in a VariablesArray into a DoubleMatrix whereby the items in the matrix are the design variables.

Private Attributes

EvaluatorCreator * _theEvalCreator

A pointer to an EvaluatorCreator used to create the evaluator used by JEGA in Dakota (a JEGAEvaluator).

JEGA::Utilities::ParameterDatabase * _theParamDB

A pointer to the ParameterDatabase from which all parameters are retrieved by the created algorithms.

VariablesArray _initPts

An array of initial points to use as an initial population.

Additional Inherited Members

13.71.1 Detailed Description

A version of Dakota::Optimizer for instantiation of John Eddy's Genetic Algorithms (JEGA).

This class encapsulates the necessary functionality for creating and properly initializing the JEGA algorithms (MOGA and SOGA).

13.71.2 Constructor & Destructor Documentation

JEGAOptimizer ( ProblemDescDB & problem_db, Model & model )

Constructs a JEGAOptimizer class object.

This method does some of the initialization work for the algorithm. In particular, it initialized the JEGA core.

Parameters

| problem_db | The Dakota::ProblemDescDB with information on how the algorithm controls should be set. |
| model | The Dakota::Model that will be used by this optimizer for problem information, etc. |

References JEGAOptimizer::theEvalCreator, ProblemDescDB::get_int(), ProblemDescDB::get_short(), Iterator::iteratedModel, JEGAOptimizer::LoadTheParameterDatabase(), Iterator::maxEvalConcurrency, Iterator::methodName, Iterator::numFinalSolutions, and Iterator::probDescDB.
13.71.3 Member Function Documentation

void LoadDakotaResponses ( const JEGA::Utilities::Design & from, Dakota::Variables & vars, Dakota::Response & resp ) const [protected]

Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects. This version is meant for the case where a Variables and a Response object exist and just need to be loaded.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>The JEGA Design class object from which to extract the variable and response information for Dakota.</td>
</tr>
<tr>
<td>vars</td>
<td>The Dakota::Variables object into which to load the design variable values of from.</td>
</tr>
<tr>
<td>resp</td>
<td>The Dakota::Response object into which to load the objective function and constraint values of from.</td>
</tr>
</tbody>
</table>

References Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Variables::discrete_string_variable(), Response::function_values(), and Dakota::set_index_to_value().

void LoadTheParameterDatabase ( ) [protected]

Reads information out of the known Dakota::ProblemDescDB and puts it into the current parameter database. This should be called from the JEGAOptimizer constructor since it is the only time when the problem description database is certain to be configured to supply data for this optimizer.

Referenced by JEGAOptimizer().

void LoadAlgorithmConfig ( JEGA::FrontEnd::AlgorithmConfig & aConfig ) [protected]

Completely initializes the supplied algorithm configuration. This loads the supplied configuration object with appropriate data retrieved from the parameter database.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aConfig</td>
<td>The algorithm configuration object to load.</td>
</tr>
</tbody>
</table>

void LoadProblemConfig ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]

Completely initializes the supplied problem configuration. This loads the fresh configuration object using the LoadTheDesignVariables, LoadTheObjectiveFunctions, and LoadTheConstraints methods.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConfig</td>
<td>The problem configuration object to load.</td>
</tr>
</tbody>
</table>

void LoadTheDesignVariables ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]

Adds DesignVariableInfo objects into the problem configuration object. This retrieves design variable information from the ParameterDatabase and creates DesignVariableInfo’s from it.
Parameters

| pConfig | The problem configuration object to load. |

References Model::continuous_lower_bounds().

void LoadTheObjectiveFunctions ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]

Adds ObjectiveFunctionInfo objects into the problem configuration object.
This retrieves objective function information from the ParameterDatabase and creates ObjectiveFunctionInfo’s from it.
Parameters

| pConfig | The problem configuration object to load. |

void LoadTheConstraints ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]

Adds ConstraintInfo objects into the problem configuration object.
This retrieves constraint function information from the ParameterDatabase and creates ConstraintInfo’s from it.
Parameters

| pConfig | The problem configuration object to load. |

References Dakota::asstring(), Dakota::copy_row_vector(), and Model::nonlinear_ineq_constraint_lower_bounds().

void GetBestSolutions ( const JEGA::Utilities::DesignOFSortSet & from, const JEGA::Algorithms::GeneticAlgorithm & theGA, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap ) [protected]

Returns up to _numBest designs sorted by DAKOTA’s fitness (L2 constraint violation, then utopia or objective), taking into account the algorithm type. The front of the returned map can be viewed as a single ”best”.
Parameters

| from | The full set of designs returned by the solver. |
| theGA | The GA used to generate this set; needed for its weights in the SO case, provided to both for consistency |
| designSortMap | Map of best solutions with key pair<constraintViolation, fitness> |

eventually this functionality must be moved into a separate post-processing application for MO datasets.

void GetBestMOSolutions ( const JEGA::Utilities::DesignOFSortSet & from, const JEGA::Algorithms::GeneticAlgorithm & theGA, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap ) [protected]

Retreive the best Designs from a set of solutions assuming that they are generated by a multi objective algorithm. eventually this functionality must be moved into a separate post-processing application for MO datasets.

void GetBestSOSolutions ( const JEGA::Utilities::DesignOFSortSet & from, const JEGA::Algorithms::GeneticAlgorithm & theGA, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap ) [protected]

Retreive the best Designs from a set of solutions assuming that they are generated by a single objective algorithm. eventually this functionality must be moved into a separate post-processing application for MO datasets.
References Dakota::abort_handler().

**JEGA::DoubleMatrix ToDoubleMatrix** ( const VariablesArray & variables ) const [protected]

Converts the items in a VariablesArray into a DoubleMatrix whereby the items in the matrix are the design variables.

The matrix will not contain responses but when being used by Dakota, this doesn’t matter. JEGA will attempt to re-evaluate these points but Dakota will recognize that they do not require re-evaluation and thus it will be a cheap operation.

**Parameters**

| variables             | The array of DakotaVariables objects to use as the contents of the returned matrix. |

**Returns**

The matrix created using the supplied VariablesArray.

**void core_run ( ) [virtual]**

Performs the iterations to determine the optimal set of solutions.

Override of pure virtual method in Optimizer base class.

The extraction of parameter values actually occurs in this method when the JEGA::FrontEnd::Driver::Execute-Algorithm is called. Also the loading of the problem and algorithm configurations occurs in this method. That way, if it is called more than once and the algorithm or problem has changed, it will be accounted for.

Reimplemented from Iterator.

References JEGAOptimizer::Driver::DestroyAlgorithm(), JEGAOptimizer::Driver::ExtractAllData(), and JEGAOptimizer::Driver::PerformIterations().

**bool accepts_multiple_points ( ) const [virtual]**

Overridden to return true since JEGA algorithms can accept multiple initial points.

**Returns**

true, always.

Reimplemented from Iterator.

**bool returns_multiple_points ( ) const [virtual]**

Overridden to return true since JEGA algorithms can return multiple final points.

**Returns**

true, always.

Reimplemented from Iterator.

**void initial_points ( const VariablesArray & pts ) [virtual]**

Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables.
**Parameters**

<table>
<thead>
<tr>
<th>pts</th>
<th>The array of initial points for the JEGA algorithm created and run by this JEGAOptimizer.</th>
</tr>
</thead>
</table>

Reimplemented from Iterator.

```cpp
const VariablesArray & initial_points ( ) const  [virtual]
```

Overridden to return the collection of initial points for the JEGA algorithm created and run by this JEGAOptimizer.

Returns

The collection of initial points for the JEGA algorithm created and run by this JEGAOptimizer.

Reimplemented from Iterator.

### 13.71.4 Member Data Documentation

**VariablesArray initPts [private]**

An array of initial points to use as an initial population.

This member is here to help support the use of JEGA algorithms in Dakota strategies. If this array is populated, then whatever initializer is specified will be ignored and the DoubleMatrix initializer will be used instead on a matrix created from the data in this array.

The documentation for this class was generated from the following files:

- JEGAOptimizer.hpp
- JEGAOptimizer.cpp

### 13.72 JEGATraits Class Reference

A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).

Inheritance diagram for JEGATraits:

```
TraitsBase
   ↓
JEGATraits
```

**Public Member Functions**

- JEGATraits ()
  
  *default constructor*

- virtual ~JEGATraits ()
  
  *destructor*

- virtual bool is_derived ()
  
  *A temporary query used in the refactor.*

- bool supports_continuous_variables ()
Return the flag indicating whether method supports continuous variables.

- bool supports_discrete_variables()

Return the flag indicating whether method supports continuous variables.

- bool supports_linear_equality()

Return the flag indicating whether method supports linear equalities.

- bool supports_linear_inequality()

Return the flag indicating whether method supports linear inequalities.

- bool supports_nonlinear_equality()

Return the flag indicating whether method supports nonlinear equalities.

- bool supports_nonlinear_inequality()

Return the flag indicating whether method supports nonlinear inequalities.

- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format()

Return the format used for nonlinear inequality constraints.

### 13.72.1 Detailed Description

A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).

The documentation for this class was generated from the following file:

- JEGAOptimizer.hpp

### 13.73 LabelsWriter Class Reference

Utility used in derived write_core to write labels in tabular format.

#### Public Member Functions

- template<typename ArrayType>
  
  void operator() (std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)

#### 13.73.1 Detailed Description

Utility used in derived write_core to write labels in tabular format.

#### 13.73.2 Member Function Documentation

void operator() ( std::ostream & s, size_t start_index, size_t num_items, const ArrayType & array_data, StringMultiArrayConstView label_array ) [inline]

The tabular labels writer only forwards the label arrays.

The documentation for this class was generated from the following file:

- DakotaVariables.hpp
13.74  **LeastSq Class Reference**

Base class for the nonlinear least squares branch of the iterator hierarchy.

Inheritance diagram for LeastSq:

```
                     Iterator
                     |
                     V
Minimizer
                     |
                     V
LeastSq
```

### Protected Member Functions

- `LeastSq (std::shared_ptr<TraitsBase> traits)`  
  default constructor
- `LeastSq (ProblemDescDB &problem_db, Model &model, std::shared_ptr<TraitsBase> traits)`  
  standard constructor
- `LeastSq (unsigned short method_name, Model &model, std::shared_ptr<TraitsBase> traits)`  
  alternate "on the fly" constructor
- `~LeastSq ()`  
  destructor
- `void initialize_run ()`
- `void post_run (std::ostream &s)`
- `void finalize_run ()`
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- `void print_results (std::ostream &s, short results_state=FINAL_RESULTS)`
- `void get_confidence_intervals (const Variables &native_vars, const Response &iter_resp)`  
  Calculate confidence intervals on estimated parameters.

### Protected Attributes

- `size_t numLeastSqTerms`  
  number of least squares terms
- `LeastSq * prevLSqInstance`  
  pointer containing previous value of leastSqInstance
- `bool weightFlag`  
  flag indicating whether weighted least squares is active
- `RealVector confBoundsLower`  
  lower bounds for confidence intervals on calibration parameters
- `RealVector confBoundsUpper`  
  upper bounds for confidence intervals on calibration parameters
- `RealVector bestIterPriFns`
CHAPTER 13. CLASS DOCUMENTATION

storage for iterator best primary functions (which shouldn’t be stored in bestResponseArray when there are transformations)

- bool retrievedIterPriFns

whether final primary iterator space functions have been retrieved (possibly by a derived class)

Static Protected Attributes

- static LeastSq * leastSqInstance

pointer to LeastSq instance used in static member functions

Private Member Functions

- void weight_model ()

Wrap iteratedModel in a RecastModel that weights the residuals.

Static Private Member Functions

- static void primary_resp_weighter (const Variables &unweighted_vars, const Variables &weighted_vars, const Response &unweighted_response, Response &weighted_response)

Recast callback function to weight least squares residuals, gradients, and Hessians.

Additional Inherited Members

13.74.1 Detailed Description

Base class for the nonlinear least squares branch of the iterator hierarchy.

The LeastSq class provides common data and functionality for least squares solvers (including NL2OL, NLSSOLLeastSq, and SNLLLeastSq.

13.74.2 Constructor & Destructor Documentation

LeastSq ( ProblemDescDB & problem_db, Model & model, std::shared_ptr<TraitsBase> traits )

[protected]

standard constructor

This constructor extracts the inherited data for the least squares branch and performs sanity checking on gradient and constraint settings.

References Dakota::abort_handler(), Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, Variables::copy(), Model::current_variables(), Minimizer::data_transform_model(), Iterator::iteratedModel, Iterator::method_Name, LeastSq::numLeastSqTerms, Minimizer::numTotalCalibTerms, Minimizer::optimizationFlag, Model::primary_fn_type(), Minimizer::scale_model(), Minimizer::scaleFlag, LeastSq::weight_model(), and LeastSq::weightFlag.

13.74.3 Member Function Documentation

void initialize_run ( ) [protected], [virtual]

This function should be invoked (or reimplemented) by any derived implementations of initialize_run() (which would otherwise hide it).

Reimplemented from Iterator.
References LeastSq::bestIterPriFns, Minimizer::initialize_run, Iterator::iteratedModel, LeastSq::leastSqInstance, Iterator::myModelLayers, LeastSq::prevLSqInstance, LeastSq::retrievedIterPriFns, and Model::update_from_subordinate_model.

```cpp
void post_run ( std::ostream & s ) [protected], [virtual]
```

Implements portions of post_run specific to LeastSq for scaling back to native variables and functions. This function should be invoked (or reimplemented) by any derived implementations of post_run() (which would otherwise hide it).

Reimplemented from Iterator.

References Dakota::abort_handler(), Iterator::activeSet, LeastSq::bestIterPriFns, Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, Variables::continuous_variables(), Model::continuous_variables(), Response::copy(), Variables::copy(), Dakota::copy_data_partial(), Model::current_response(), ScalingModel::cv_scaled2native(), Model::db_lookup(), Model::evaluate(), Response::function_gradients(), Response::function_values_view(), LeastSq::get_confidence_intervals(), Iterator::iteratedModel, Model::model_rep(), Minimizer::numContinuousVars, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, Minimizer::original_model(), Iterator::outputLevel, Minimizer::post_run(), ActiveSet::request_value(), ActiveSet::request_values(), ActiveSet::request_vector(), LeastSq::retrievedIterPriFns, Minimizer::scaleFlag, Minimizer::scalingModel, Minimizer::vendorNumericalGradFlag, and LeastSq::weightFlag.

```cpp
void finalize_run ( ) [inline], [protected], [virtual]
```

Utility function to perform common operations following post_run(); deallocation and resetting of instance pointers. Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.

References Minimizer::finalize_run(), LeastSq::leastSqInstance, and LeastSq::prevLSqInstance.

```cpp
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
```

Redefines default iterator results printing to include nonlinear least squares results (residual terms and constraints).

Reimplemented from Iterator.

References Iterator::activeSet, Minimizer::archive_allocate_best, Minimizer::archive_best, Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, LeastSq::confBoundsLower, LeastSq::confBoundsUpper, ExperimentData::config_vars(), Model::continuous_variable_labels(), Dakota::data_pairs, Minimizer::dataTransformModel, Minimizer::expData, Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Model::model_rep(), ExperimentData::num_experiments(), Model::num_functions(), Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, Minimizer::original_model(), DataTransformModel::print_best_responses(), Minimizer::print_residuals(), ActiveSet::request_values(), Minimizer::scaleFlag, LeastSq::weightFlag, and Dakota::write_precision.

```cpp
void get_confidence_intervals ( const Variables & native_vars, const Response & iter_resp ) [protected]
```

Calculate confidence intervals on estimated parameters.

Calculate individual confidence intervals for each parameter, based on a linear approximation of the nonlinear model. native_cv are needed for transformations and final reporting. iter_resp must contain the final differenced, scaled, weighted residuals and gradients.
References LeastSq::confBoundsLower, LeastSq::confBoundsUpper, Variables::continuous_variables(), Response::copy(), Response::function_gradients_view(), Response::function_values(), Model::model_rep(), Minimizer::numContinuousVars, LeastSq::numLeastSqTerms, ScalingModel::response_modify_s2n(), Minimizer::scaleFlag, Minimizer::scalingModel, and Minimizer::vendorNumericalGradFlag.

Referenced by LeastSq::post_run().

```cpp
void weight_model ( ) [private]
```

Wrap iteratedModel in a RecastModel that weights the residuals.

Setup Recast for weighting model. The weighting transformation doesn’t resize, so use numUserPrimaryFns. No vars, active set or secondary mapping. All indices are one-to-one mapped (no change in counts).

References Dakota::abort_handler(), Model::assign_rep(), Model::current_response(), Response::function_gradients(), Response::function_hessians(), Iterator::iteratedModel, Iterator::myModelLayers, Minimizer::numContinuousVars, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Iterator::outputLevel, LeastSq::primary_resp_weighter(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and Model::subordinate_model().

Referenced by LeastSq::LeastSq().

```cpp
void primary_resp_weighter ( const Variables & unweighted vars, const Variables & weighted vars, const Response & unweighted response, Response & weighted response ) [static], [private]
```

Recast callback function to weight least squares residuals, gradients, and Hessians.

Apply weights to least squares residuals

References Dakota::NPOS, Response::active_set_derivative_vector(), Response::active_set_request_vector(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::continuous_variable_ids(), Variables::cv(), Dakota::find_index(), Response::function_gradients(), Response::function_gradients_view(), Response::function_hessian(), Response::function_hessian_view(), Response::function_values(), Response::function_values_view(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Iterator::iteratedModel, LeastSq::leastSqInstance, LeastSq::numLeastSqTerms, Iterator::outputLevel, Model::primary_response_fn_weights(), and Model::subordinate_model().

Referenced by LeastSq::weight_model().

The documentation for this class was generated from the following files:

- DakotaLeastSq.hpp
- DakotaLeastSq.cpp

### 13.75 LibraryEnvironment Class Reference

**Environment** corresponding to execution as an embedded library.

Inheritance diagram for LibraryEnvironment:

```
Environment
```

```
LibraryEnvironment
```

```
Public Member Functions

- **LibraryEnvironment ()**
  
  Default constructor

- **LibraryEnvironment (ProgramOptions prog_opts, bool check_bcast_construct=true, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)**
  
  Primary constructor: program options typically specifies an input file or input string. Optionally specify a callback function to be invoked after parsing. Set check_bcast_construct if performing late updates and later calling done_modifying_db().

- **LibraryEnvironment (MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts=ProgramOptions(), bool check_bcast_construct=true, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)**
  
  Alternate constructor accepting communicator; same options as primary.

- **~LibraryEnvironment ()**
  
  Destructor

- **void insert_nodes (Dakota::DataMethod &dme, Dakota::DataModel &dmo, Dakota::DataVariables &dv, Dakota::DataInterface &di, Dakota::DataResponses &dr)**
  
  Insert DB nodes for a {Method,Model,Variables,Interface,Responses} set.

- **void done_modifying_db ()**
  
  Check database contents, broadcast, and construct iterators.

- **bool plugin_interface (const String &model_typ, const String &interf_typ, const String &an_driver, Interface *plugin_iface)**
  
  Plug-in the passed interface into any interface matching the specified (possibly empty) model, interface, and driver strings; returns true if a plugin was performed.

- **InterfaceList filtered_interface_list (const String &interf_typ, const String &an_driver)**
  
  Filter the available Interface instances based on matching interface type and analysis drivers (empty String matches any)

- **ModelList filtered_model_list (const String &model_typ, const String &interf_typ, const String &an_driver)**
  
  Filter the available Model instances based on matching model type, interface type, and analysis drivers (empty String matches any)

Additional Inherited Members

### 13.75.1 Detailed Description

**Environment** corresponding to execution as an embedded library.

This environment corresponds to use of Dakota as a library within another application, e.g., within library_mode.cpp. It sets up the ParallelLibrary and ProblemDescDB objects without access to command line arguments.

### 13.75.2 Constructor & Destructor Documentation

**LibraryEnvironment ( ProgramOptions prog_opts, bool check_bcast_construct = true, DbCallbackFunctionPtr callback = NULL, void *callback_data = NULL )**

Primary constructor: program options typically specifies an input file or input string. Optionally specify a callback function to be invoked after parsing. Set check_bcast_construct if performing late updates and later calling done_modifying_db().

Construct library environment, optionally performing check/bcast of database and iterator construction

References Environment::construct(), OutputManager::output_startup_message(), Environment::outputManager, and Environment::parse().
LibraryEnvironment ( MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts = ProgramOptions (),
bool check_bcast_construct = true, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL )

Alternate constructor accepting communicator, same options as primary.
Construct library environment on passed MPI Comm, optionally performing check/bcast of database and
iterator construction. MPI Comm is first argument so client doesn’t have to pass all args
References Environment::construct(), OutputManager::output_startup_message(), Environment::outputManager,
and Environment::parse().

### 13.75.3 Member Function Documentation

**InterfaceList filtered_interface_list ( const String & interf_type, const String & an_driver )**

filter the available Interface instances based on matching interface type and analysis drivers (empty String matches any)
This convenience function helps clients locate and plugin to the right Interface instance for simple cases. Pass an empty string to match any instead of a specific instance
References Interface::analysis_drivers(), Dakota::contains(), Interface::interface_type(), ProblemDescDB::model_list(), and Environment::probDescDB.

**ModelList filtered_model_list ( const String & model_type, const String & interf_type, const String & an_driver )**

filter the available Model instances based on matching model type, interface type, and analysis drivers (empty String matches any)
This convenience function helps clients locate and plugin to the right Interface instance for cases where the parallel configuration is needed in constructing a parallel plugin. Pass an empty string to match any instead of a specific instance
References Interface::analysis_drivers(), Dakota::contains(), Interface::interface_type(), ProblemDescDB::model_list(), and Environment::probDescDB.
Referenced by parallel_interface_plugin(), LibraryEnvironment::plugin_interface(), run_dakota(), and run_dakota_mixed().
The documentation for this class was generated from the following files:

- LibraryEnvironment.hpp
- LibraryEnvironment.cpp

### 13.76 LightWtBaseConstructor Struct Reference

Dummy struct for overloading constructors used in on-the-fly Model instantiations.

**Public Member Functions**

- LightWtBaseConstructor (int=0)
  
  C++ structs can have constructors.
13.76.1 Detailed Description

Dummy struct for overloading constructors used in on-the-fly Model instantiations. LightWtBaseConstructor is used to overload the constructor used for on-the-fly Model instantiations. Putting this struct here avoids circular dependencies.

The documentation for this struct was generated from the following file:

- dakota.global_defs.hpp

13.77 MatchesWC Struct Reference

Predicate that returns true when the passed path matches the wild_card with which it was configured. Currently supports * and ?.

**Public Member Functions**

- MatchesWC (const bfs::path &wild_card)
  
  ctor that builds and stores the regular expression

- bool operator() (const bfs::path &dir_entry)
  
  return true is dir_entry matches wildCardRegEx

**Public Attributes**

- boost::basic_regex
  
  < bfs::path::value_type > wildCardRegEx
  
  archived RegEx; wchar-based on Windows

13.77.1 Detailed Description

Predicate that returns true when the passed path matches the wild_card with which it was configured. Currently supports * and ?.

The documentation for this struct was generated from the following file:

- WorkdirHelper.hpp

13.78 MatlabInterface Class Reference

Inheritance diagram for MatlabInterface:

```
+-------------------+                       +-------------------+  +-------------------+
| Interface          |                       | ApplicationInterface |  | MatlabInterface    |
|                   |                       |                     |  |                   |
+-------------------+                       +-------------------+  +-------------------+
|                   |                       | DirectApplicInterface |  |                   |
+-------------------+                       +-------------------+  +-------------------+
```


Public Member Functions

- MatlabInterface (const ProblemDescDB &problem_db)
  - Constructor: start Matlab engine.
- ~MatlabInterface ()
  - Destructor: close Matlab engine.

Protected Member Functions

- virtual int derived_map_ac (const String &ac_name)
  - execute an analysis code portion of a direct evaluation invocation
- int matlab_engine_run (const Dakota::String &ac_name)
  - Helper function supporting derived_map_ac. Sends data to Matlab, executes analysis, collects return data.
- int matlab_field_prep (mxArray *dakota_matlab, const char *field_name)
  - check that the dakota_matlab structure has the specified field_name and add if necessary; free structure memory in preparation for new alloc

Protected Attributes

- engine * matlabEngine
  - pointer to the MATLAB engine used for direct evaluations

13.78.1 Detailed Description

Specialization of DirectApplicInterface to link to Matlab analysis drivers. Includes convenience functions to map data to/from Matlab

13.78.2 Member Function Documentation

int derived_map_ac (const String & ac_name) [protected], [virtual]

execute an analysis code portion of a direct evaluation invocation
Matlab specialization of derived analysis components.
Reimplemented from DirectApplicInterface.
References ApplicationInterface::analysisServerId, and MatlabInterface::matlab_engine_run().

int matlab_engine_run (const Dakota::String & ac_name) [protected]

Helper function supporting derived_map_ac. Sends data to Matlab, executes analysis, collects return data.
Direct interface to Matlab through Mathworks external API. m-file executed is specified through analysis_drivers, extra strings through analysis_components. (Original BMA 11/28/2005)
Special thanks to Lee Peterson for substantial enhancements 12/15/2007: Added output buffer for the M-ATLAB command response and error messages Made the Dakota variable persistent in the MATLAB engine workspace Added robustness to the user deleting required Dakota fields
References Dakota::abort_handler(), DirectApplicInterface::analysisComponents, DirectApplicInterface::analysisDriverIndex, Interface::currEvalId, DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, Dakota::FIELD_NAMES, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, Interface::fnLabels, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, MatlabInterface::matlab_field_prep(), MatlabInterface::matlabEngine, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, Dakota::NUMBER_OF_FIELD, DirectApplicInterface::numDerivVars,
DirectApplicInterface::numFns, DirectApplicInterface::numVars, Interface::outputLevel, DirectApplicInterface-::xC, DirectApplicInterface::xCLabels, DirectApplicInterface::xDI, DirectApplicInterface::xDILabels, DirectApplicInterface::xDR, and DirectApplicInterface::xDRLabels.

Referenced by MatlabInterface::derived_map_ac().

The documentation for this class was generated from the following files:

- MatlabInterface.hpp
- MatlabInterface.cpp

### 13.79 MetaIterator Class Reference

Base class for meta-iterators.

Inheritance diagram for MetaIterator:

```
MetaIterator
  
  Iterator

  CollabHybridMetaIterator
  ConcurrentMetaIterator
  EmbedHybridMetaIterator
  SeqHybridMetaIterator
```

#### Public Member Functions

- `bool resize()`  
  reinitializes iterator based on new variable size

#### Protected Member Functions

- `MetaIterator (ProblemDescDB &problem_db)`  
  standard constructor

- `MetaIterator (ProblemDescDB &problem_db, Model &model)`  
  alternate constructor

- `~MetaIterator()`  
  destructor

- `void post_run (std::ostream &s)`  
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/-Responses and perform final analysis phase in a standalone way

- `void check_model (const String &method_ptr, const String &model_ptr)`  
  check that a model identified by pointer has the same id as the iteratedModel passed through the ctor chain

- `void allocate_by_pointer (const String &method_ptr, Iterator &the_iterator, Model &the_model)`  
  initialize the_iterator and the_model based on method_ptr

- `void allocate_by_name (const String &method_string, const String &model_ptr, Iterator &the_iterator, Model &the_model)`  
  initialize the_iterator based on method_string

- `std::pair<int, int> estimate_by_pointer (const String &method_ptr, Iterator &the_iterator, Model &the_-
  model)`
**CHAPTER 13. CLASS DOCUMENTATION**

`estimate minimum and maximum processors per iterator needed for init_iterator_parallelism(); instantiates the_iterator and the_model as needed, but on minimal processor ranks (is later augmented by allocate_by_pointer())`

- std::pair<int, int> `estimate_by_name` (const String &method_string, const String &model_ptr, Iterator &the_iterator, Model &the_model)

`estimate minimum and maximum processors per iterator needed for init_iterator_parallelism(); instantiates the_iterator and the_model as needed, but on minimal processor ranks (is later augmented by allocate_by_name())`

**Protected Attributes**

- IteratorScheduler `iterSched`  
  scheduler for concurrent execution of Iterators
- int `maxIteratorConcurrency`  
  maximum number of concurrent sub-iterator executions

**Additional Inherited Members**

**13.79.1 Detailed Description**

Base class for meta-iterators. This base class shares code for concurrent and hybrid meta-iterators, where the former supports multi-start and Pareto set iteration and the latter supports sequential, embedded, and collaborative hybrids.

**13.79.2 Member Function Documentation**

```cpp
void post_run ( std::ostream & s ) [protected], [virtual]
```

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way.

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `post_run()`, typically after performing its own implementation steps.

Reimplemented from `Iterator`.

References MetaIterator::iterSched, IteratorScheduler::lead_rank(), and Iterator::print_results().

The documentation for this class was generated from the following files:

- MetaIterator.hpp
- MetaIterator.cpp

**13.80 Minimizer Class Reference**

Base class for the optimizer and least squares branches of the iterator hierarchy.

Inheritance diagram for Minimizer:

```plaintext
Minimizer
| +-- LeastSquares
|    | +-- LEACoOptimator
|    | +-- BoundedLEACoOptimator
|    | +-- NLP/FSOptimizer
|    | +-- OptimizerOptimizer
|    | +-- ESGetAutoMinimizer
|    | +-- SurfBasedLocalMinimizer
|    | +-- SurfBasedGlobalMinimizer
```

Inheritance diagram for `Minimizer`:

- `Minimizer`
  - `LeastSquares`
    - `LEACoOptimator`
    - `BoundedLEACoOptimator`
    - `NLP/FSOptimizer`
    - `OptimizerOptimizer`
    - `ESGetAutoMinimizer`
    - `SurfBasedLocalMinimizer`
    - `SurfBasedGlobalMinimizer`
Public Member Functions

- void constraint_tolerance (Real const_tol)
  set the method constraint tolerance (constraintTol)
- Real constraint_tolerance () const
  return the method constraint tolerance (constraintTol)
- bool resize ()
  reinitializes iterator based on new variable size

Static Public Member Functions

- static Real sum_squared_residuals (size_t num_pri_fns, const RealVector &residuals, const RealVector &weights)
  return weighted sum of squared residuals
- static void print_residuals (size_t num_terms, const RealVector &best_terms, const RealVector &weights, size_t num_best, size_t best_index, std::ostream &s)
  print num_terms residuals and misfit for final results
- static void print_model_resp (size_t num_pri_fns, const RealVector &best_fns, size_t num_best, size_t best_index, std::ostream &s)
  print the original user model resp in the case of data transformations

Protected Member Functions

- Minimizer (std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  default constructor
- Minimizer (ProblemDescDB &problem_db, Model &model, std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  standard constructor
- Minimizer (unsigned short method_name, Model &model, std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  alternate constructor for "on the fly" instantiations
- Minimizer (unsigned short method_name, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq, std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  alternate constructor for "on the fly" instantiations
- ~Minimizer ()
  destructor
- void update_from_model (const Model &model)
  set inherited data attributes based on extractions from incoming model
- void initialize_run ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
- void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- const Model & algorithm_space_model () const
CHAPTER 13. CLASS DOCUMENTATION

- **Model original** (unsigned short recasts_left=0)
  
  Return a shallow copy of the original model this Iterator was originally passed, optionally leaving recasts_left on top of it.

- **void data_transform_model ()**
  
  Wrap iteratedModel in a RecastModel that subtracts provided observed data from the primary response functions (variables and secondary responses are unchanged)

- **void scale_model ()**
  
  Wrap iteratedModel in a RecastModel that performs variable and/or response scaling.

- **Real objective (const RealVector &fn_vals, const BoolDeque &max_sense, const RealVector &primary_wts) const**
  
  compute a composite objective value from one or more primary functions

- **Real objective (const RealVector &fn_vals, size_t num_fns, const BoolDeque &max_sense, const RealVector &primary_wts) const**
  
  compute a composite objective with specified number of source primary functions, instead of userPrimaryFns

- **void objective_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &max_sense, const RealVector &primary_wts, RealVector &obj_grad) const**
  
  compute the gradient of the composite objective function

- **void objective_gradient (const RealVector &fn_vals, size_t num_fns, const RealMatrix &fn_grads, const BoolDeque &max_sense, const RealVector &primary_wts, RealVector &obj_grad) const**
  
  compute the gradient of the composite objective function

- **void objective_hessian (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &max_sense, const RealVector &primary_wts, RealSymMatrix &obj_hess) const**
  
  compute the Hessian of the composite objective function

- **void objective_hessian (const RealVector &fn_vals, size_t num_fns, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &max_sense, const RealVector &primary_wts, RealSymMatrix &obj_hess) const**
  
  compute the Hessian of the composite objective function

- **void archive_allocate_best (size_t num_points)**
  
  allocate results arrays and labels for multipoint storage

- **void archive_best (size_t index, const Variables &best_vars, const Response &best_resp)**
  
  archive the best point into the results array

- **void resize_best_vars_array (size_t newsize)**
  
  Safely resize the best variables array to newsize taking into account the envelope-letter design pattern and any recasting.

- **void resize_best_resp_array (size_t newsize)**
  
  Safely resize the best response array to newsize taking into account the envelope-letter design pattern and any recasting.

- **void local_recast_retrieve (const Variables &vars, Response &response) const**
  
  infers MOO/NLS solution from the solution of a single-objective optimizer
Protected Attributes

- `size_t numFunctions`
  number of response functions
- `size_t numContinuousVars`
  number of active continuous vars
- `size_t numDiscreteIntVars`
  number of active discrete integer vars
- `size_t numDiscreteStringVars`
  number of active discrete string vars
- `size_t numDiscreteRealVars`
  number of active discrete real vars
- `Real constraintTol`
  optimizer/least squares constraint tolerance
- `Real bigRealBoundSize`
  cutoff value for inequality constraint and continuous variable bounds
- `int bigIntBoundSize`
  cutoff value for discrete variable bounds
- `size_t numNonlinearIneqConstraints`
  number of nonlinear inequality constraints
- `size_t numNonlinearEqConstraints`
  number of nonlinear equality constraints
- `size_t numLinearIneqConstraints`
  number of linear inequality constraints
- `size_t numLinearEqConstraints`
  number of linear equality constraints
- `size_t numNonlinearConstraints`
  total number of nonlinear constraints
- `size_t numLinearConstraints`
  total number of linear constraints
- `size_t numConstraints`
  total number of linear and nonlinear constraints
- `bool optimizationFlag`
  flag for use where optimization and NLS must be distinguished
- `size_t numUserPrimaryFns`
  number of objective functions or least squares terms in the inbound model; always initialize at Minimizer, even if overridden later
- `size_t numIterPrimaryFns`
  number of objective functions or least squares terms in iterator’s view, after transformations; always initialize at Minimizer, even if overridden later
- `bool boundConstraintFlag`
  convenience flag for denoting the presence of user-specified bound constraints. Used for method selection and error checking.
- `bool speculativeFlag`
flag for speculative gradient evaluations

- bool calibrationDataFlag
  
  flag indicating whether user-supplied calibration data is active

- ExperimentData expData
  
  Container for experimental data to which to calibrate model using least squares or other formulations which minimize SSE.

- size_t numExperiments
  
  number of experiments

- size_t numTotalCalibTerms
  
  number of total calibration terms (sum over experiments of number of experimental data per experiment, including field data)

- Model dataTransformModel
  
  Shallow copy of the data transformation model, when present (cached in case further wrapped by other transformations)

- bool scaleFlag
  
  whether Iterator-level scaling is active

- Model scalingModel
  
  Shallow copy of the scaling transformation model, when present (cached in case further wrapped by other transformations)

- Minimizer * prevMinInstance
  
  pointer containing previous value of minimizerInstance

- bool vendorNumericalGradFlag
  
  convenience flag for gradient_type == numerical && method_source == vendor

Static Protected Attributes

- static Minimizer * minimizerInstance
  
  pointer to Minimizer used in static member functions

Friends

- class SOLBase
  
  the SOLBase class is not derived the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)

- class SNLLBase
  
  the SNLLBase class is not derived the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)

Additional Inherited Members

13.80.1 Detailed Description

Base class for the optimizer and least squares branches of the iterator hierarchy.

The Minimizer class provides common data and functionality for Optimizer and LeastSq.
13.80.2 Constructor & Destructor Documentation

Minimizer (ProblemDescDB & problem_db, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()), [protected]

standard constructor

This constructor extracts inherited data for the optimizer and least squares branches and performs sanity checking on constraint settings.

References Iterator::iteratedModel, Iterator::maxIterations, Iterator::methodName, Iterator::numFinalSolutions, and Minimizer::updateFromModel().

13.80.3 Member Function Documentation

void initialize_run ( ) [protected], [virtual]

utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in Optimizer.
References Model::all_continuous_variables(), Model::all_discrete_int_variables(), Model::all_discrete_real_variables(), Iterator::bestVariablesArray, Model::initialize_mapping(), Model::is_null(), Iterator::iteratedModel, Model::mapping_initialized(), Iterator::methodPCIter, Minimizer::minimizerInstance, Iterator::myModelLayers, Minimizer::prevMinInstance, Minimizer::resize(), Model::set_evaluation_reference(), Iterator::subIteratorFlag, Model::subordinate_model(), and Iterator::summaryOutputFlag.

Referenced by LeastSq::initialize_run(), and Optimizer::initialize_run().

void post_run ( std::ostream & s ) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in Optimizer, and SurrBasedLocalMinimizer.
References Model::is_null(), Iterator::iteratedModel, Model::print_evaluation_summary(), Iterator::print_results(), Iterator::resultsDB, Iterator::summaryOutputFlag, and ResultsManager::write_databases().

Referenced by LeastSq::post_run(), SurrBasedLocalMinimizer::post_run(), and Optimizer::post_run().

void finalize_run ( ) [protected], [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in Optimizer.
References Model::finalize_mapping(), Iterator::finalize_run(), Iterator::iteratedModel, Model::mapping_initialized(), Minimizer::minimizerInstance, Minimizer::prevMinInstance, and Minimizer::resize().

Referenced by LeastSq::finalize_run(), and Optimizer::finalize_run().

```cpp
const Model & algorithm::space_model() const [inline], [protected], [virtual]
```

default definition that gets redefined in selected derived Minimizers

Reimplemented from Iterator.

Reimplemented in EffGlobalMinimizer.

References Iterator::iteratedModel.

```cpp
void data_transform_model() [protected]
```

Wrap iteratedModel in a RecastModel that subtracts provided observed data from the primary response functions (variables and secondary responses are unchanged)

Reads observation data to compute least squares residuals. Does not change size of responses, and is the first wrapper, therefore sizes are based on iteratedModel.

References Dakota::abort_handler(), Iterator::activeSet, Iterator::assign_rep(), Minimizer::dataTransformModel, Minimizer::expData, ProblemDescDB::get_sizet(), Iterator::iteratedModel, ExperimentData::load_data(), Iterator::myModelLayers, Model::num_functions(), Model::num_primary_fns(), Minimizer::numExperiments, Minimizer::numFunctions, Minimizer::numIterPrimaryFns, Minimizer::numTotalCalibTerms, Iterator::outputLevel, Iterator::probDescDB, and ActiveSet::request_vector().

Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

```cpp
void scale_model() [protected]
```

Wrap iteratedModel in a RecastModel that performs variable and/or response scaling.

Wrap the iteratedModel in a scaling transformation, such that iteratedModel now contains a scaling recast model. Potentially affects variables, primary, and secondary responses

References Model::assign_rep(), Iterator::iteratedModel, Iterator::myModelLayers, and Minimizer::scalingModel.

Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

```cpp
Real objective ( const RealVector & fn_vals, const BoolDeque & max_sense, const RealVector & primary_wts ) const [protected]
```

compute a composite objective value from one or more primary functions

The composite objective computation sums up the contributions from one of more primary functions using the primary response fn weights.

References Minimizer::numUserPrimaryFns.

Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), SurrBasedMinimizer::augmented_lagrangian_merit(), EffGlobalMinimizer::expected_improvement(), SurrBasedMinimizer::initialize_filter(), SurrBasedMinimizer::lagrangian_merit(), Optimizer::objective_reduction(), SurrBasedMinimizer::penalty_merit(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

```cpp
Real objective ( const RealVector & fn_vals, size_t num_fns, const BoolDeque & max_sense, const RealVector & primary_wts ) const [protected]
```

compute a composite objective with specified number of source primary functions, instead of userPrimaryFns
This “composite” objective is a more general case of the previous objective(), but doesn’t presume a reduction map from user to iterated space. Used to apply weights and sense in COLIN results sorting. Leaving as a duplicate implementation pending resolution of COLIN lookups.

References Minimizer::optimizationFlag.

```cpp
void objective_gradient ( const RealVector & fn_vals, size_t num_fns, const RealMatrix & fn_grads, const BoolDeque & max_sense, const RealVector & primary_wts, RealVector & obj_grad ) const
[protected]
```

compute the gradient of the composite objective function

The composite objective gradient computation combines the contributions from one of more primary function gradients, including the effect of any primary function weights. In the case of a linear mapping (MOO), only the primary function gradients are required, but in the case of a nonlinear mapping (NLS), primary function values are also needed. Within RecastModel::set_mapping(), the active set requests are automatically augmented to make values available when needed, based on nonlinearRespMapping settings.

References Minimizer::numContinuousVars, and Minimizer::optimizationFlag.

```cpp
void objective_hessian ( const RealVector & fn_vals, size_t num_fns, const RealMatrix & fn_grads, const RealSymMatrixArray & fn_hessians, const BoolDeque & max_sense, const RealVector & primary_wts, RealSymMatrix & obj_hess ) const [protected]
```

compute the Hessian of the composite objective function

The composite objective Hessian computation combines the contributions from one of more primary function Hessians, including the effect of any primary function weights. In the case of a linear mapping (MOO), only the primary function Hessians are required, but in the case of a nonlinear mapping (NLS), primary function values and gradients are also needed in general (gradients only in the case of a Gauss-Newton approximation). Within the default RecastModel::set_mapping(), the active set requests are automatically augmented to make values and gradients available when needed, based on nonlinearRespMapping settings.

References Dakota::abort_handler(), Minimizer::numContinuousVars, and Minimizer::optimizationFlag.

```cpp
void resize_best_vars_array ( size_t newsize ) [protected]
```

Safely resize the best variables array to newsize taking into account the envelope-letter design pattern and any recasting.

Uses data from the innermost model, should any Minimizer recasts be active. Called by multipoint return solvers. Do not directly call resize on the bestVariablesArray object unless you intend to share the internal content (letter) with other objects after assignment.

References Iterator::bestVariablesArray, Variables::copy(), Model::current_variables(), and Minimizer::original_model().

```cpp
void resize_best_resps_array ( size_t newsize ) [protected]
```

Safely resize the best response array to newsize taking into account the envelope-letter design pattern and any recasting.

Uses data from the innermost model, should any Minimizer recasts be active. Called by multipoint return solvers. Do not directly call resize on the bestResponseArray object unless you intend to share the internal content (letter) with other objects after assignment.

References Iterator::bestResponseArray, Response::copy(), Model::current_response(), and Minimizer::original_model().
void local_recast_retrieve ( const Variables & vars, Response & response ) const  [protected]

infers MOO/NLS solution from the solution of a single-objective optimizer

Retrieve a MOO/NLS response based on the data returned by a single objective optimizer by performing a
data_pairs search. This may get called even for a single user-specified function, since we may be recasting a single
NLS residual into a squared objective. Always returns best data in the space of the original inbound Model.
References Response::active_set(), Dakota::data_pairs, Model::interface_id(), Iterator::iteratedModel, Dakota-
::lookup_by_val(), and Response::update().
Referenced by Optimizer::post_run().
The documentation for this class was generated from the following files:
- DakotaMinimizer.hpp
- DakotaMinimizer.cpp

13.81 MixedVarConstraints Class Reference

Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type
array merging).
Inheritance diagram for MixedVarConstraints:

```
<table>
<thead>
<tr>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>MixedVarConstraints</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- MixedVarConstraints (const SharedVariablesData & svd)
  lightweight constructor
- MixedVarConstraints (const ProblemDescDB & problem_db, const SharedVariablesData & svd)
  standard constructor
- ~MixedVarConstraints ()
  destructor
- void write (std::ostream &s) const
  write a variable constraints object to an std::ostream
- void read (std::istream &s)
  read a variable constraints object from an std::istream

Additional Inherited Members

13.81.1 Detailed Description

Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type
array merging).

Derived variable constraints classes take different views of the design, uncertain, and state variable types
and the continuous and discrete domain types. The MixedVarConstraints derived class separates the continuous
and discrete domain types (see Variables::get_variables(problem_db) for variables type selection; variables type is
passed to the Constraints constructor in Model).
13.82. MIXEDVARIABLES CLASS REFERENCE

13.81.2 Constructor & Destructor Documentation

MixedVarConstraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )

standard constructor

In this class, mixed continuous/discrete variables are used. Most iterators/strategies use this approach, which is the default in Constraints::get_constraints().

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_mv(), Constraints::sharedVarsData, and SharedVariablesData::view().

The documentation for this class was generated from the following files:

- MixedVarConstraints.hpp
- MixedVarConstraints.cpp

13.82 MixedVariables Class Reference

Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

Inheritance diagram for MixedVariables:

```
Variables

MixedVariables
```

Public Member Functions

- MixedVariables (const ProblemDescDB &problem_db, const std::pair< short, short > &view)
  standard constructor
- MixedVariables (const SharedVariablesData &svd)
  lightweight constructor
- ~MixedVariables ()
  destructor

Protected Member Functions

- void read (std::istream &s)
  read a variables object from an std::istream
- void write (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  write a variables object to an std::ostream, e.g., the console, optionally specifying which partition (all/active/inactive)
- void write_aprepro (std::ostream &s) const
  write a variables object to an std::ostream in aprepro format, e.g., a parameters file
- void read_tabular (std::istream &s, unsigned short vars_part=ALL_VARS)
- void write_tabular (std::ostream &s, unsigned short vars_part=ALL_VARS) const
CHAPTER 13. CLASS DOCUMENTATION

write a variables object in tabular format to an std::ostream, optionally specifying which partition (all/active/inactive)

- void write_tabular_labels (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  write the labels in input spec order to a std::ostream, optionally specifying which partition (all/active/inactive)

- template<typename Reader>
  void read_core (std::istream &s, Reader read_handler, unsigned short vars_part)
  Implementation of reading various formats using the specified read handler.

- template<typename Writer>
  void write_core (std::ostream &s, Writer write_handler, unsigned short vars_part) const
  Implementation of writing various formats using the specified write handler.

Additional Inherited Members

13.82.1 Detailed Description

Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MixedVariables derived class separates the continuous and discrete domain types (see Variables::get_variables(problem_db)).

13.82.2 Constructor & Destructor Documentation

MixedVariables ( const ProblemDescDB & problem_db, const std::pair<short, short> & view )
standard constructor

In this class, the distinct approach is used (design, uncertain, and state variable types and continuous and discrete domain types are distinct). Most iterators/strategies use this approach.

References Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteStringVars, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), ProblemDescDB::get_sa(), Variables::sharedVarsData, and SharedVariablesData::view().

13.82.3 Member Function Documentation

void read_tabular ( std::istream & s, unsigned short vars_part = ALL_VARS ) [protected], [virtual]
Tabular reader that reads data in order design, aleatory, epistemic, state according to counts in vc_totals (extract in order: cdv/ddiv/ddrv, cauv/dauiv/daurv, ceuv/deuiv/deurv, csv/dsiv/dsrv, which might reflect active or all depending on context. Assumes container sized, since might be a view into a larger array.
Reimplemented from Variables.
References MixedVariables::read_core().

void read_core ( std::istream & s, Reader read_handler, unsigned short vars_part ) [protected]
Implementation of reading various formats using the specified read handler.
Reordering is required in all read/write cases that will be visible to the user since all derived vars classes should use the same CDV/DDV/UV/CSV/DSV ordering for clarity. Neutral file I/O, binary streams, and packed buffers do not need to reorder (so long as read/write are consistent) since this data is not intended for public consumption.
References SharedVariablesData::active_components_totals(), Variables::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Variables::all_discrete_string_variable_labels(), Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteStringVars, SharedVariablesData::components_totals(), SharedVariablesData::cv_start(), SharedVariablesData::dv_start(), SharedVariablesData::drv_start(), SharedVariablesData::dsv_start(), SharedVariablesData::icv_start(), SharedVariablesData::idiv_start(), SharedVariablesData::idrv_start(), SharedVariablesData::idsv_start(), SharedVariablesData::inactive_components_totals(), and Variables::sharedVarsData.

Referenced by MixedVariables::read(), and MixedVariables::read_tabular().

The documentation for this class was generated from the following files:

- MixedVariables.hpp
- MixedVariables.cpp

13.83 Model Class Reference

Base class for the model class hierarchy.

Inheritance diagram for Model:

```
Model
  NestedModel RecastModel SimulationModel SurrogateModel
  ActiveSubspaceModel
  AdaptedBasisModel
  DataTransformModel
  ProbabilityTransformModel
  RandomFieldModel
  ScalingModel
  DataFitSurrModel
  HierarchSurrModel
```

Public Member Functions

- **Model ()**
  
  *default constructor*

- **Model (ProblemDescDB &problem_db)**
  
  *standard constructor for envelope*

- **Model (const Model &model)**
  
  *copy constructor*

- **virtual ~Model ()**
  
  *destructor*

- **Model operator= (const Model &model)**
  
  *assignment operator*

- **virtual Iterator & subordinate_iterator ()**
  
  *return the sub-iterator in nested and surrogate models*

- **virtual Model & subordinate_model ()**
virtual Model & surrogate_model ()

return the active approximation sub-model in surrogate models

virtual void surrogate_model_indices (size_t lf_model_index, size_t lf_soln_lev_index=NPOS)

set the indices that define the active approximation sub-model within surrogate models

virtual void surrogate_model_indices (const SizetSizetPair &lf_form_level)

set the index pair that defines the active approximation sub-model within surrogate models

virtual const SizetSizetPair & surrogate_model_indices () const

return the indices of the active approximation sub-model within surrogate models

virtual Model & truth_model ()

return the active truth sub-model in surrogate models

virtual void truth_model_indices (size_t hf_model_index, size_t hf_soln_lev_index=NPOS)

set the indices that define the active truth sub-model within surrogate models

virtual void truth_model_indices (const SizetSizetPair &hf_form_level)

set the index pair that defines the active truth sub-model within surrogate models

virtual const SizetSizetPair & truth_model_indices () const

return the indices of the active truth sub-model within surrogate models

virtual void derived_subordinate_models (ModelList &ml, bool recurse_flag)

portion of subordinate_models() specific to derived model classes

virtual void update_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())

propagate vars/labels/bounds/targets from the bottom up

virtual Interface & derived_interface ()

return the interface employed by the derived model class, if present: SimulationModel::userDefinedInterface, DataFitSurrModel::approxInterface, or NestedModel::optionalInterface

virtual size_t solution_levels (bool lwr_bnd=true) const

number of discrete levels within solution control (SimulationModel)

virtual void solution_level_index (size_t index)

activate a particular level within the solution level control and return the cost estimate (SimulationModel)

virtual size_t solution_level_index () const

return currently active level within the solution level control (SimulationModel)

virtual RealVector solution_level_costs () const

return ordered cost estimates across solution levels (SimulationModel)

virtual Real solution_level_cost () const

return currently active cost estimate from solution level control (SimulationModel)

virtual void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)

set the relative weightings for multiple objective functions or least squares terms

virtual void surrogate_function_indices (const IntSet &surr_fn_indices)

set the (currently active) surrogate function index set

virtual bool initialize_mapping (ParLevLIter pl_iter)

initialize model mapping, returns true if the variables size has changed

virtual bool finalize_mapping ()

finalize model mapping, returns true if the variables size has changed
• virtual bool mapping_initialized () const
  return true if mapping has been fully initialized, false otherwise.

• virtual bool resize_pending () const
  return true if a potential resize is still pending, such that sizing-based initialization should be deferred.

• virtual void build_approximation (size_t index=_NPOS)
  build a new SurrogateModel approximation.

• virtual bool build_approximation (const Variables &vars, const IntResponsePair &response_pr, size_t index=_NPOS)
  build a new SurrogateModel approximation using/enforcing anchor response at vars; rebuild if needed.

• virtual void update_approximation (bool rebuild_flag, size_t index=_NPOS)
  replace the approximation data within an existing surrogate based on data updates propagated elsewhere.

• virtual void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag, size_t index=_NPOS)
  replace the anchor point data within an existing surrogate.

• virtual void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  replace the data points within an existing surrogate.

• virtual void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  replace the data points within an existing surrogate.

• virtual void append_approximation (bool rebuild_flag, size_t index=_NPOS)
  append to the existing approximation data within a surrogate based on data updates propagated elsewhere.

• virtual void append_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag, size_t index=_NPOS)
  append a single point to an existing surrogate’s data.

• virtual void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  append multiple points to an existing surrogate’s data.

• virtual void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=_NPOS)
  append multiple points to an existing surrogate’s data.

• virtual void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
  remove the previous data set addition to a surrogate (e.g., due to a previous append_approximation() call); flag manages storing of surrogate data for use in a subsequent push_approximation().

• virtual void push_approximation ()
  push a previous approximation data state; reverse of pop_approximation.

• virtual bool push_available ()
  query for whether a trial increment is restorable within a surrogate.

• virtual void finalize_approximation ()
  finalize an approximation by applying all previous trial increments.

• virtual void store_approximation (size_t index=_NPOS)
  move the current approximation into storage for later combination; the index of the stored set can be passed to allow replacement instead of augmentation (default is push_back).

• virtual void restore_approximation (size_t index=_NPOS)
return an approximation from storage; the index identifies a particular stored data set (default is pop_back from stored)

- virtual void remove_stored_approximation (size_t index=_NPOS)
  remove a stored approximation, due to redundancy with the current approximation, prior to combination (default for no index is pop_back)

- virtual void combine_approximation ()
  combine the current approximation with previously stored data sets

- virtual void clear_stored ()
  clear stored approximations

- virtual void run_dace_iterator (bool rebuild_flag, size_t index=_NPOS)
  execute the DACE iterator, append the approximation data, and rebuild the approximation if indicated

- virtual bool force_rebuild ()
  determine whether a surrogate model rebuild should be forced based on changes in the inactive data

- virtual SharedApproxData & shared_approximation ()
  retrieve the shared approximation data within the ApproximationInterface of a DataFitSurrModel

- virtual std::vector<Approximation> & approximations ()
  retrieve the set of Approximations within the ApproximationInterface of a DataFitSurrModel

- virtual const Pecos::SurrogateData & approximation_data (size_t index)
  retrieve the approximation data from a particular Approximation instance within the ApproximationInterface of a DataFitSurrModel

- virtual const RealVectorArray & approximation_coefficients (bool normalized=false)
  retrieve the approximation coefficients from each Approximation within a DataFitSurrModel

- virtual void approximation_coefficients (const RealVectorArray &approx_coeffs, bool normalized=false)
  set the approximation coefficients for each Approximation within a DataFitSurrModel

- virtual const RealVector & approximation_variances (const Variables &vars)
  retrieve the prediction variances from each Approximation within a DataFitSurrModel

- virtual void surrogate_response_mode (short mode)
  set response computation mode used in SurrogateModels for forming currentResponse

- virtual short surrogate_response_mode () const
  return response computation mode used in SurrogateModels for forming currentResponse

- virtual const RealVector & error_estimates ()
  retrieve error estimates corresponding to the Model’s response (could be surrogate error for SurrogateModels, statistical MSE for NestedModels, or adjoint error estimates for SimulationModels). Errors returned correspond to most recent evaluate().

- virtual DiscrepancyCorrection & discrepancy_correction ()
  return the DiscrepancyCorrection object used by SurrogateModels

- virtual short correction_type ()
  return the correction type from the DiscrepancyCorrection object used by SurrogateModels

- virtual void single_apply (const Variables &vars, Response &resp, const SizetSizet2DPair &indices)
  apply the DiscrepancyCorrection object to correct an approximation within a SurrogateModel

- virtual void recursive_apply (const Variables &vars, Response &resp)
  apply the DiscrepancyCorrection object to recursively correct an approximation within a HierarchSurrModel
• virtual void \texttt{component\_parallel\_mode} (short mode)
  
  update component\_Parallel\_Mode for supporting parallelism in model sub-components
• virtual \texttt{IntIntPair estimate\_partition\_bounds} (int max\_eval\_concurrency)
  
  estimate the minimum and maximum partition sizes that can be utilized by this \texttt{Model}
• virtual size\_t \texttt{mi\_parallel\_level\_index} () const
  
  return the index for the metaiterator-iterator parallelism level within ParallelConfiguration::\texttt{miPLIters} that is active for use in a particular \texttt{Model} at runtime
• virtual void \texttt{cache\_unmatched\_response} (int raw\_id)
  
  migrate an unmatched response record from active response map (computed by synchronize() or synchronize\_nowait()) to cached response map
• virtual short \texttt{local\_eval\_synchronization} ()
  
  return derived model synchronization setting
• virtual int \texttt{local\_eval\_concurrency} ()
  
  return derived model asynchronous evaluation concurrency
• virtual void \texttt{serve\_run} (ParLevLIter pl Iter, int max\_eval\_concurrency)
  
  Service job requests received from the master. Completes when a termination message is received from stop\_servers().
• virtual void \texttt{stop\_servers} ()
  
  Executed by the master to terminate all server operations for a particular model when iteration on the model is complete.
• virtual bool \texttt{derived\_master\_overload} () const
  
  Return a flag indicating the combination of multiprocessor evaluations and a dedicated master iterator scheduling. Used in synchronous evaluate functions to prevent the error of trying to run a multiprocessor job on the master.
• virtual void \texttt{inactive\_view} (short view, bool recurse\_flag=true)
  
  update the Model’s inactive view based on higher level (nested) context
• virtual const String & \texttt{interface\_id} () const
  
  return the interface identifier
• virtual int \texttt{derived\_evaluation\_id} () const
  
  Return the value of the evaluation id counter for the Model.
• virtual bool \texttt{evaluation\_cache} (bool recurse\_flag=true) const
  
  Indicates the usage of an evaluation cache by the Model.
• virtual bool \texttt{restart\_file} (bool recurse\_flag=true) const
  
  Indicates the usage of a restart file by the Model.
• virtual void \texttt{set\_evaluation\_reference} ()
  
  Set the reference points for the evaluation counters within the Model.
• virtual void \texttt{fine\_grained\_evaluation\_counters} ()
  
  Request fine-grained evaluation reporting within the Model.
• virtual void \texttt{print\_evaluation\_summary} (std::ostream &s, bool minimal\_header=false, bool relative\_count=true) const
  
  Print an evaluation summary for the Model.
• virtual void \texttt{eval\_tag\_prefix} (const String &eval\_id\_str)
  
  set the hierarchical eval ID tag prefix
• virtual bool \texttt{db\_lookup} (const Variables &search\_vars, const ActiveSet &search\_set, Response &found\_resp)
search the eval database (during derivative estimation); derived may need to reimplement due to problem transformations \textit{(RecastModel)}; return true if found in DB

- virtual void \texttt{stop}\textunderscore init\textunderscore mapping (ParLevLIter pl\_iter)
  
called from \texttt{IteratorScheduler::run\_iterator()} for iteratorComm rank 0 to terminate \texttt{serve\_init\_mapping()} on other iteratorComm processors

- virtual int \texttt{serve}\textunderscore init\textunderscore mapping (ParLevLIter pl\_iter)
  
called from \texttt{IteratorScheduler::run\_iterator()} for iteratorComm rank != 0 to balance resize() calls on iteratorComm rank 0

- virtual void \texttt{stop}\textunderscore finalize\textunderscore mapping (ParLevLIter pl\_iter)
  
called from \texttt{IteratorScheduler::run\_iterator()} for iteratorComm rank 0 to terminate \texttt{serve\_finalize\_mapping()} on other iteratorComm processors

- virtual int \texttt{serve}\textunderscore finalize\textunderscore mapping (ParLevLIter pl\_iter)
  
called from \texttt{IteratorScheduler::run\_iterator()} for iteratorComm rank != 0 to balance resize() calls on iteratorComm rank 0

- virtual void \texttt{warm}\textunderscore start\textunderscore flag (const bool flag)
  
set the warm start flag (warmStartFlag)

- ModelList & \texttt{subordinate}\textunderscore models (bool recurse\_flag=true)
  
return the sub-models in nested and surrogate models

- void \texttt{evaluate} ()
  
Compute the Response at currentVariables (default ActiveSet).

- void \texttt{evaluate} (const ActiveSet &set)
  
Compute the Response at currentVariables (specified ActiveSet).

- void \texttt{evaluate\_nowait} ()
  
Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (default ActiveSet).

- void \texttt{evaluate\_nowait} (const ActiveSet &set)
  
Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (specified ActiveSet).

- const IntResponseMap & \texttt{synchronize} ()
  
Execute a blocking scheduling algorithm to collect the complete set of results from a group of asynchronous evaluations.

- const IntResponseMap & \texttt{synchronize\_nowait} ()
  
Execute a nonblocking scheduling algorithm to collect all available results from a group of asynchronous evaluations.

- int \texttt{evaluation\_id} () const
  
return Model’s (top-level) evaluation counter, not to be confused with derived counter returned by \texttt{derived\_evaluation\_id()}

- void \texttt{init}\textunderscore communicators (ParLevLIter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag=true)
  
allocate communicator partitions for a model and store configuration in modelPCIterMap

- void \texttt{init}\textunderscore serial ()
  
for cases where \texttt{init}\textunderscore communicators() will not be called, modify some default settings to behave properly in serial.

- void \texttt{set}\textunderscore communicators (ParLevLIter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag=true)
  
set active parallel configuration for the model (set modelPCIter from modelPCIterMap)

- void \texttt{free}\textunderscore communicators (ParLevLIter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag=true)
  
deallocate communicator partitions for a model
• MPIComm `analysis_comm () const`
  retrieve the MPI communicator on which this model is configured to conduct function evaluation analyses (provided for library clients)

• void `stop_init_communicators (ParLevLIter pl_iter)`
  called from `IteratorScheduler::init_iterator()` for iteratorComm rank 0 to terminate `serve_init_communicators()` on other iteratorComm processors

• int `serve_init_communicators (ParLevLIter pl_iter)`
  called from `IteratorScheduler::init_iterator()` for iteratorComm rank != 0 to balance `init_communicators()` calls on iteratorComm rank 0

• void `estimate_message_lengths ()`
  estimate messageLengths for a model

• bool `manage_data_recastings ()`
  initialize modelList and recastFlags for data import/export

• bool `recastings () const`
  return true if recastFlags is defined

• void `user_space_to_iterator_space (const Variables &user_vars, const Response &user_resp, Variables &iter_vars, Response &iter_resp)`
  employ the model recursion to transform from bottom level user-space data to top level iterator-space data

• void `iterator_space_to_user_space (const Variables &iter_vars, const Response &iter_resp, Variables &user_vars, Response &user_resp)`
  employ the model recursion to transform from top level iterator-space data to bottom level user-space data

• void `assign_rep (Model *model_rep, bool ref_count_incr=true)`
  replaces existing letter with a new one

• size_t `tv () const`
  returns total number of vars

• size_t `cv () const`
  returns number of active continuous variables

• size_t `div () const`
  returns number of active discrete integer vars

• size_t `dsv () const`
  returns number of active discrete string vars

• size_t `drv () const`
  returns number of active discrete real vars

• size_t `icv () const`
  returns number of inactive continuous variables

• size_t `idiv () const`
  returns number of inactive discrete integer vars

• size_t `idsv () const`
  returns number of inactive discrete string vars

• size_t `idrv () const`
  returns number of inactive discrete real vars

• size_t `acv () const`
  returns total number of continuous variables

• size_t `adiv () const`
returns total number of discrete integer vars
  • `size_t adsv () const`
returns total number of discrete string vars
  • `size_t adrv () const`
returns total number of discrete real vars
  • `void active_variables (const Variables &vars)`
set the active variables in currentVariables
  • `const RealVector & continuous_variables () const`
return the active continuous variables from currentVariables
  • `Real continuous_variable (size_t i) const`
return an active continuous variable from currentVariables
  • `void continuous_variables (const RealVector &c_vars)`
set the active continuous variables in currentVariables
  • `void continuous_variable (Real c_var, size_t i)`
set an active continuous variable in currentVariables
  • `const IntVector & discrete_int_variables () const`
return the active discrete integer variables from currentVariables
  • `int discrete_int_variable (size_t i) const`
return an active discrete integer variable from currentVariables
  • `void discrete_int_variables (const IntVector &d_vars)`
set the active discrete integer variables in currentVariables
  • `void discrete_int_variable (int d_var, size_t i)`
set an active discrete integer variable in currentVariables
  • `StringMultiArrayConstView discrete_string_variables () const`
return the active discrete string variables from currentVariables
  • `const String & discrete_string_variable (size_t i) const`
return an active discrete string variable from currentVariables
  • `void discrete_string_variables (StringMultiArrayConstView d_vars)`
set the active discrete string variables in currentVariables
  • `void discrete_string_variable (const String &d_var, size_t i)`
set an active discrete string variable in currentVariables
  • `const RealVector & discrete_real_variables () const`
return the active discrete real variables from currentVariables
  • `Real discrete_real_variable (size_t i) const`
return an active discrete real variable from currentVariables
  • `void discrete_real_variables (const RealVector &d_vars)`
set the active discrete real variables in currentVariables
  • `void discrete_real_variable (Real d_var, size_t i)`
set an active discrete real variable in currentVariables
  • `UShortMultiArrayConstView continuous_variable_types () const`
return the active continuous variable types from currentVariables
  • `void continuous_variable_types (UShortMultiArrayConstView cv_types)`
set the active continuous variable types in currentVariables
• void continuous_variable_type (unsigned short cv_type, size_t i)
  set an active continuous variable type in currentVariables
• UShortMultiArrayConstView discrete_int_variable_types () const
  return the active discrete variable types from currentVariables
• void discrete_int_variable_types (UShortMultiArrayConstView div_types)
  set the active discrete variable types in currentVariables
• void discrete_int_variable_type (unsigned short div_type, size_t i)
  set an active discrete variable type in currentVariables
• UShortMultiArrayConstView discrete_string_variable_types () const
  return the active discrete variable types from currentVariables
• void discrete_string_variable_types (UShortMultiArrayConstView div_types)
  set the active discrete variable types in currentVariables
• void discrete_string_variable_type (unsigned short div_type, size_t i)
  set an active discrete variable type in currentVariables
• UShortMultiArrayConstView discrete_real_variable_types () const
  return the active discrete variable types from currentVariables
• void discrete_real_variable_types (UShortMultiArrayConstView drv_types)
  set the active discrete variable types in currentVariables
• void discrete_real_variable_type (unsigned short drv_type, size_t i)
  set an active discrete variable type in currentVariables
• SizetMultiArrayConstView continuous_variable_ids () const
  return the active continuous variable identifiers from currentVariables
• void continuous_variable_ids (SizetMultiArrayConstView cv_ids)
  set the active continuous variable identifiers in currentVariables
• void continuous_variable_id (size_t cv_id, size_t i)
  set an active continuous variable identifier in currentVariables
• const RealVector & inactive_continuous_variables () const
  return the inactive continuous variables in currentVariables
• void inactive_continuous_variables (const RealVector &i_c_vars)
  set the inactive continuous variables in currentVariables
• const IntVector & inactive_discrete_int_variables () const
  return the inactive discrete variables in currentVariables
• void inactive_discrete_int_variables (const IntVector &i_d_vars)
  set the inactive discrete variables in currentVariables
• StringMultiArrayConstView inactive_discrete_string_variables () const
  return the inactive discrete variables in currentVariables
• void inactive_discrete_string_variables (StringMultiArrayConstView i_d_vars)
  set the inactive discrete variables in currentVariables
• const RealVector & inactive_discrete_real_variables () const
  return the inactive discrete variables in currentVariables
• void inactive_discrete_real_variables (const RealVector &i_d_vars)
  set the inactive discrete variables in currentVariables
• UShortMultiArrayConstView inactive_continuous_variable_types () const
return the inactive continuous variable types from currentVariables

- SizetMultiArrayConstView inactive_continuous_variable_ids () const
to return the inactive continuous variable identifiers from currentVariables

- const RealVector & all_continuous_variables () const
to return all continuous variables in currentVariables

- void all_continuous_variables (const RealVector &a_c-vars)
to set all continuous variables in currentVariables

- void all_continuous_variable (Real a_c_var, size_t i)
to set a variable within the all continuous variables in currentVariables

- const IntVector & all_discrete_int_variables () const
to return all discrete variables in currentVariables

- void all_discrete_int_variables (const IntVector &a_d-vars)
to set all discrete variables in currentVariables

- void all_discrete_int_variable (int a_d_var, size_t i)
to set a variable within the all discrete variables in currentVariables

- StringMultiArrayConstView all_discrete_string_variables () const
to return all discrete variables in currentVariables

- void all_discrete_string_variables (StringMultiArrayConstView a_d-vars)
to set all discrete variables in currentVariables

- void all_discrete_string_variable (const String &a_d_var, size_t i)
to set a variable within the all discrete variables in currentVariables

- const RealVector & all_discrete_real_variables () const
to return all discrete variables in currentVariables

- void all_discrete_real_variables (const RealVector &a_d-vars)
to set all discrete variables in currentVariables

- void all_discrete_real_variable (Real a_d_var, size_t i)
to set a variable within the all discrete variables in currentVariables

- UShortMultiArrayConstView all_continuous_variable_types () const
to return all continuous variable types from currentVariables

- UShortMultiArrayConstView all_discrete_int_variable_types () const
to return all discrete variable types from currentVariables

- UShortMultiArrayConstView all_discrete_string_variable_types () const
to return all discrete variable types from currentVariables

- UShortMultiArrayConstView all_discrete_real_variable_types () const
to return all discrete variable types from currentVariables

- SizetMultiArrayConstView all_continuous_variable_ids () const
to return all continuous variable identifiers from currentVariables

- const IntSetArray & discrete_design_set_int_values () const
to return the sets of values available for each of the discrete design set integer variables

- void discrete_design_set_int_values (const IntSetArray &isa)
to define the sets of values available for each of the discrete design set integer variables

- const StringSetArray & discrete_design_set_string_values () const
to return the sets of values available for each of the discrete design set string variables
• **void** discrete_design_set_string_values (const StringSetArray &ssa)  
  *define the sets of values available for each of the discrete design set string variables*

• **const RealSetArray &** discrete_design_set_real_values () const  
  *return the sets of values available for each of the discrete design set real variables*

• **void** discrete_design_set_real_values (const RealSetArray &rsa)  
  *define the sets of values available for each of the discrete design set real variables*

• **const IntSetArray &** discrete_state_set_int_values () const  
  *return the sets of values available for each of the discrete state set integer variables*

• **void** discrete_state_set_int_values (const IntSetArray &isa)  
  *define the sets of values available for each of the discrete state set integer variables*

• **const StringSetArray &** discrete_state_set_string_values () const  
  *return the sets of values available for each of the discrete state set string variables*

• **void** discrete_state_set_string_values (const StringSetArray &ssa)  
  *define the sets of values available for each of the discrete state set string variables*

• **const RealSetArray &** discrete_state_set_real_values () const  
  *return the sets of values available for each of the discrete state set real variables*

• **void** discrete_state_set_real_values (const RealSetArray &rsa)  
  *define the sets of values available for each of the discrete state set real variables*

• **const BitArray & discrete_int_sets ()**  
  *define and return discreteIntSets using active view from currentVariables*

• **const BitArray & discrete_int_sets (short active_view)**  
  *define and return discreteIntSets using passed active view*

• **const IntSetArray & discrete_set_int_values ()**  
  *return the sets of values available for each of the active discrete set integer variables (aggregated in activeDiscSet-IntValues)*

• **const IntSetArray & discrete_set_int_values (short active_view)**  
  *return the sets of values available for each of the active discrete set integer variables (aggregated in activeDiscSet-IntValues)*

• **const StringSetArray & discrete_set_string_values ()**  
  *return the sets of values available for each of the active discrete set string variables (aggregated in activeDiscSet-StringValues)*

• **const StringSetArray & discrete_set_string_values (short active_view)**  
  *return the sets of values available for each of the active discrete set string variables (aggregated in activeDiscSet-StringValues)*

• **const RealSetArray & discrete_set_real_values ()**  
  *return the sets of values available for each of the active discrete set real variables (aggregated in activeDiscSet-RealValues)*

• **const RealSetArray & discrete_set_real_values (short active_view)**  
  *return the sets of values available for each of the active discrete set real variables (aggregated in activeDiscSet-RealValues)*

• **Pecos::AleatoryDistParams & aleatory_distribution_parameters ()**  
  *return aleatDistParams*

• **const Pecos::AleatoryDistParams & aleatory_distribution_parameters () const**  
  *return aleatDistParams*
• void aleatory_distribution_parameters (const Pecos::AleatoryDistParams &adp)
  set aleatDistParams
• Pecos::EpistemicDistParams & epistemic_distribution_parameters ()
  return epistDistParams
• const Pecos::EpistemicDistParams & epistemic_distribution_parameters () const
  return epistDistParams
• void epistemic_distribution_parameters (const Pecos::EpistemicDistParams &edp)
  set epistDistParams
• StringMultiArrayConstView continuous_variable_labels () const
  return the active continuous variable labels from currentVariables
• void continuous_variable_labels (StringMultiArrayConstView c_v_labels)
  set the active continuous variable labels in currentVariables
• StringMultiArrayConstView discrete_int_variable_labels () const
  return the active discrete variable labels from currentVariables
• void discrete_int_variable_labels (StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables
• StringMultiArrayConstView discrete_string_variable_labels () const
  return the active discrete variable labels from currentVariables
• void discrete_string_variable_labels (StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables
• StringMultiArrayConstView discrete_real_variable_labels () const
  return the active discrete variable labels from currentVariables
• void discrete_real_variable_labels (StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables
• StringMultiArrayConstView inactive_continuous_variable_labels () const
  return the inactive continuous variable labels in currentVariables
• void inactive_continuous_variable_labels (StringMultiArrayConstView i_c_v_labels)
  set the inactive continuous variable labels in currentVariables
• StringMultiArrayConstView inactive_discrete_int_variable_labels () const
  return the inactive discrete variable labels in currentVariables
• void inactive_discrete_int_variable_labels (StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables
• StringMultiArrayConstView inactive_discrete_string_variable_labels () const
  return the inactive discrete variable labels in currentVariables
• void inactive_discrete_string_variable_labels (StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables
• StringMultiArrayConstView inactive_discrete_real_variable_labels () const
  return the inactive discrete variable labels in currentVariables
• void inactive_discrete_real_variable_labels (StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables
• StringMultiArrayConstView all_continuous_variable_labels () const
  return all continuous variable labels in currentVariables
• void all_continuous_variable_labels (StringMultiArrayConstView a_c_v_labels)
set all continuous variable labels in currentVariables

- **void all_continuous_variable_labels** (const String &a_c_v_label, size_t i)

  set a label within the all continuous labels in currentVariables

- **StringMultiArrayConstView all_discrete_int_variable_labels** () const

  return all discrete variable labels in currentVariables

- **void all_discrete_int_variable_labels** (StringMultiArrayConstView a_d_v_labels)

  set all discrete variable labels in currentVariables

- **void all_discrete_int_variable_label** (const String &a_d_v_label, size_t i)

  set a label within the all discrete labels in currentVariables

- **StringMultiArrayConstView all_discrete_string_variable_labels** () const

  return all discrete variable labels in currentVariables

- **void all_discrete_string_variable_labels** (StringMultiArrayConstView a_d_v_labels)

  set all discrete variable labels in currentVariables

- **void all_discrete_string_variable_label** (const String &a_d_v_label, size_t i)

  set a label within the all discrete labels in currentVariables

- **StringMultiArrayConstView all_discrete_real_variable_labels** () const

  return all discrete variable labels in currentVariables

- **void all_discrete_real_variable_labels** (StringMultiArrayConstView a_d_v_labels)

  set all discrete variable labels in currentVariables

- **void all_discrete_real_variable_label** (const String &a_d_v_label, size_t i)

  set a label within the all discrete labels in currentVariables

- **const StringArray & response_labels** () const

  return the response labels from currentResponse

- **void response_labels** (const StringArray &resp_labels)

  set the response labels in currentResponse

- **const RealVector & continuous_lower_bounds** () const

  return the active continuous lower bounds from userDefinedConstraints

- **Real continuous_lower_bound** (size_t i) const

  return an active continuous lower bound from userDefinedConstraints

- **void continuous_lower_bounds** (const RealVector &c_l_bnds)

  set the active continuous lower bounds in userDefinedConstraints

- **void continuous_lower_bound** (Real c_l_bnd, size_t i)

  set the i-th active continuous lower bound in userDefinedConstraints

- **const RealVector & continuous_upper_bounds** () const

  return the active continuous upper bounds from userDefinedConstraints

- **Real continuous_upper_bound** (size_t i) const

  return an active continuous upper bound from userDefinedConstraints

- **void continuous_upper_bounds** (const RealVector &c_u_bnds)

  set the active continuous upper bounds in userDefinedConstraints

- **void continuous_upper_bound** (Real c_u_bnd, size_t i)

  set the i-th active continuous upper bound from userDefinedConstraints

- **const IntVector & discrete_int_lower_bounds** () const

  return the active discrete int lower bounds from userDefinedConstraints
• int discrete_int_lower_bound (size_t i) const
  return an active discrete int lower bound from userDefinedConstraints
• void discrete_int_lower_bounds (const IntVector &d_l_bnds)
  set the active discrete int lower bounds in userDefinedConstraints
• void discrete_int_lower_bound (int d_l_bnd, size_t i)
  set the i-th active discrete int lower bound in userDefinedConstraints
• const IntVector & discrete_int_upper_bounds () const
  return the active discrete int upper bounds from userDefinedConstraints
• int discrete_int_upper_bound (size_t i) const
  return an active discrete int upper bound from userDefinedConstraints
• void discrete_int_upper_bounds (const IntVector &d_u_bnds)
  set the active discrete int upper bounds in userDefinedConstraints
• void discrete_int_upper_bound (int d_u_bnd, size_t i)
  set the i-th active discrete int upper bound in userDefinedConstraints
• const RealVector & discrete_real_lower_bounds () const
  return the active discrete real lower bounds from userDefinedConstraints
• Real discrete_real_lower_bound (size_t i) const
  return an active discrete real lower bound from userDefinedConstraints
• void discrete_real_lower_bounds (const RealVector &d_l_bnds)
  set the active discrete real lower bounds in userDefinedConstraints
• void discrete_real_lower_bound (Real d_l_bnd, size_t i)
  set the i-th active discrete real lower bound in userDefinedConstraints
• const RealVector & discrete_real_upper_bounds () const
  return the active discrete real upper bounds from userDefinedConstraints
• Real discrete_real_upper_bound (size_t i) const
  return an active discrete real upper bound from userDefinedConstraints
• void discrete_real_upper_bounds (const RealVector &d_u_bnds)
  set the active discrete real upper bounds in userDefinedConstraints
• void discrete_real_upper_bound (Real d_u_bnd, size_t i)
  set the i-th active discrete real upper bound in userDefinedConstraints
• const RealVector & inactive_continuous_lower_bounds () const
  return the inactive continuous lower bounds in userDefinedConstraints
• void inactive_continuous_lower_bounds (const RealVector &i_c_l_bnds)
  set the inactive continuous lower bounds in userDefinedConstraints
• const RealVector & inactive_continuous_upper_bounds () const
  return the inactive continuous upper bounds in userDefinedConstraints
• void inactive_continuous_upper_bounds (const RealVector &i_c_u_bnds)
  set the inactive continuous upper bounds in userDefinedConstraints
• const IntVector & inactive_discrete_int_lower_bounds () const
  return the inactive discrete lower bounds in userDefinedConstraints
• void inactive_discrete_int_lower_bounds (const IntVector &i_d_l_bnds)
  set the inactive discrete lower bounds in userDefinedConstraints
• const IntVector & inactive_discrete_int_upper_bounds () const

return the inactive discrete upper bounds in userDefinedConstraints

- void inactive_discrete_int_upper_bounds (const IntVector &i_d_u_bnds)
set the inactive discrete upper bounds in userDefinedConstraints
- const RealVector & inactive_discrete_real_lower_bounds () const
return the inactive discrete lower bounds in userDefinedConstraints
- void inactive_discrete_real_lower_bounds (const RealVector &i_d_l_bnds)
set the inactive discrete lower bounds in userDefinedConstraints
- const RealVector & inactive_discrete_real_upper_bounds () const
return the inactive discrete upper bounds in userDefinedConstraints
- void inactive_discrete_real_upper_bounds (const RealVector &i_d_u_bnds)
set the inactive discrete upper bounds in userDefinedConstraints
- const RealVector & all_continuous_lower_bounds () const
return all continuous lower bounds in userDefinedConstraints
- void all_continuous_lower_bounds (const RealVector &a_c_l_bnds)
set all continuous lower bounds in userDefinedConstraints
- void all_continuous_lower_bound (Real a_c_l_bnd, size_t i)
set a lower bound within continuous lower bounds in userDefinedConstraints
- const RealVector & all_continuous_upper_bounds () const
return all continuous upper bounds in userDefinedConstraints
- void all_continuous_upper_bounds (const RealVector &a_c_u_bnds)
set all continuous upper bounds in userDefinedConstraints
- void all_continuous_upper_bound (Real a_c_u_bnd, size_t i)
set an upper bound within all continuous upper bounds in userDefinedConstraints
- const IntVector & all_discrete_int_lower_bounds () const
return all discrete lower bounds in userDefinedConstraints
- void all_discrete_int_lower_bounds (const IntVector &a_d_l_bnds)
set all discrete lower bounds in userDefinedConstraints
- void all_discrete_int_lower_bound (int a_d_l_bnd, size_t i)
set a lower bound within all discrete lower bounds in userDefinedConstraints
- const IntVector & all_discrete_int_upper_bounds () const
return all discrete upper bounds in userDefinedConstraints
- void all_discrete_int_upper_bounds (const IntVector &a_d_u_bnds)
set all discrete upper bounds in userDefinedConstraints
- void all_discrete_int_upper_bound (int a_d_u_bnd, size_t i)
set an upper bound within all discrete upper bounds in userDefinedConstraints
- const RealVector & all_discrete_real_lower_bounds () const
return all discrete lower bounds in userDefinedConstraints
- void all_discrete_real_lower_bounds (const RealVector &a_d_l_bnds)
set all discrete lower bounds in userDefinedConstraints
- void all_discrete_real_lower_bound (Real a_d_l_bnd, size_t i)
set a lower bound within all discrete lower bounds in userDefinedConstraints
- const RealVector & all_discrete_real_upper_bounds () const
return all discrete upper bounds in userDefinedConstraints
• void all_discrete_real_upper_bounds (const RealVector &a_d_u_bnds)
  set all discrete upper bounds in userDefinedConstraints
• void all_discrete_real_upper_bound (Real a_d_u_bnd, size_t i)
  set an upper bound within all discrete upper bounds in userDefinedConstraints
• size_t num_linear_ineq_constraints () const
  return the number of linear inequality constraints
• size_t num_linear_eq_constraints () const
  return the number of linear equality constraints
• const RealMatrix & linear_ineq_constraint_coeffs () const
  return the linear inequality constraint coefficients
• void linear_ineq_constraint_coeffs (const RealMatrix &lin_ineq_coeffs)
  set the linear inequality constraint coefficients
• const RealVector & linear_ineq_constraint_lower_bounds () const
  return the linear inequality constraint lower bounds
• void linear_ineq_constraint_lower_bounds (const RealVector &lin_ineq_l_bnds)
  set the linear inequality constraint lower bounds
• const RealVector & linear_ineq_constraint_upper_bounds () const
  return the linear inequality constraint upper bounds
• void linear_ineq_constraint_upper_bounds (const RealVector &lin_ineq_u_bnds)
  set the linear inequality constraint upper bounds
• const RealMatrix & linear_eq_constraint_coeffs () const
  return the linear equality constraint coefficients
• void linear_eq_constraint_coeffs (const RealMatrix &lin_eq_coeffs)
  set the linear equality constraint coefficients
• const RealVector & linear_eq_constraint_targets () const
  return the linear equality constraint targets
• void linear_eq_constraint_targets (const RealVector &lin_eq_targets)
  set the linear equality constraint targets
• size_t num_nonlinear_ineq_constraints () const
  return the number of nonlinear inequality constraints
• size_t num_nonlinear_eq_constraints () const
  return the number of nonlinear equality constraints
• const RealVector & nonlinear_ineq_constraint_lower_bounds () const
  return the nonlinear inequality constraint lower bounds
• void nonlinear_ineq_constraint_lower_bounds (const RealVector &nln_ineq_l_bnds)
  set the nonlinear inequality constraint lower bounds
• const RealVector & nonlinear_ineq_constraint_upper_bounds () const
  return the nonlinear inequality constraint upper bounds
• void nonlinear_ineq_constraint_upper_bounds (const RealVector &nln_ineq_u_bnds)
  set the nonlinear inequality constraint upper bounds
• const RealVector & nonlinear_eq_constraint_targets () const
  return the nonlinear equality constraint targets
• void nonlinear_eq_constraint_targets (const RealVector &nln_eq_targets)
set the nonlinear equality constraint targets

- const Variables & current_variables () const
  return the current variables (currentVariables) as const reference (preferred)

- Variables & current_variables ()
  return the current variables (currentVariables) in mutable form (special cases)

- const Constraints & user_defined_constraints () const
  return the user-defined constraints (userDefinedConstraints)

- const Response & current_response () const
  return the current response (currentResponse)

- ProblemDescDB & problem_description_db () const
  return the problem description database (probDescDB)

- ParallelLibrary & parallel_library () const
  return the parallel library (parallelLib)

- const String & model_type () const
  return the model type (modelType)

- const String & surrogate_type () const
  return the surrogate type (surrogateType)

- const String & model_id () const
  return the model identifier (modelId)

- size_t num_functions () const
  return number of functions in currentResponse

- size_t num_primary_fns () const
  return number of primary functions (total less nonlinear constraints)

- size_t num_secondary_fns () const
  return number of secondary functions (number of nonlinear constraints)

- const String & gradient_type () const
  return the gradient evaluation type (gradientType)

- const String & method_source () const
  return the numerical gradient evaluation method source (methodSource)

- const String & interval_type () const
  return the numerical gradient evaluation interval type (intervalType)

- bool ignore_bounds () const
  option for ignoring bounds when numerically estimating derivatives

- bool central_hess () const
  option for using old 2nd-order scheme when computing finite-diff Hessian

- const RealVector & fd_gradient_step_size () const
  return the finite difference gradient step size (fdGradStepSize)

- const String & fd_gradient_step_type () const
  return the finite difference gradient step type (fdGradStepType)

- const IntSet & gradient_id_analytic () const
  return the mixed gradient analytic IDs (gradIdAnalytic)

- const IntSet & gradient_id_numerical () const
  return the mixed gradient numerical IDs (gradIdNumerical)
• const String & hessian_type () const
  return the Hessian evaluation type (hessianType)
• const String & quasi_hessian_type () const
  return the Hessian evaluation type (quasiHessType)
• const RealVector & fd_hessian_by_grad_step_size () const
  return gradient-based finite difference Hessian step size (fdHessByGradStepSize)
• const RealVector & fd_hessian_by_fn_step_size () const
  return function-based finite difference Hessian step size (fdHessByFnStepSize)
• const String & fd_hessian_step_type () const
  return the finite difference Hessian step type (fdHessStepType)
• const IntSet & hessian_id_analytic () const
  return the mixed Hessian analytic IDs (hessIdAnalytic)
• const IntSet & hessian_id_numerical () const
  return the mixed Hessian analytic IDs (hessIdNumerical)
• const IntSet & hessian_id_quasi () const
  return the mixed Hessian analytic IDs (hessIdQuasi)
• void primary_response_fn_sense (const BoolDeque &sense)
  set the optimization sense for multiple objective functions
• const BoolDeque & primary_response_fn_sense () const
  get the optimization sense for multiple objective functions
• const RealVector & primary_response_fn_weights () const
  get the relative weightings for multiple objective functions or least squares terms
• const ScalingOptions & scaling_options () const
  user-provided scaling options
• short primary_fn_type () const
  get the primary response function type (generic, objective, calibration)
• void primary_fn_type (short type)
  set the primary response function type, e.g., when recasting
• bool derivative_estimation ()
  indicates potential usage of estimate_derivatives() based on gradientType/hessianType
• void supports_derivative_estimation (bool sed_flag)
  set whether this model should perform or pass on derivative estimation
• void init_comms_bcast_flag (bool ich_flag)
  set initCommsBcastFlag
• int evaluation_capacity () const
  return the evaluation capacity for use in iterator logic
• int derivative_concurrency () const
  return the gradient concurrency for use in parallel configuration logic
• bool asynch_flag () const
  return the asynchronous evaluation flag (asynchEvalFlag)
• void asynch_flag (const bool flag)
  set the asynchronous evaluation flag (asynchEvalFlag)
• short output_level () const
return the outputLevel
• void output_level (const short level)
  set the outputLevel
• const IntArray & message_lengths () const
  return the array of MPI packed message buffer lengths (messageLengths)
• void parallel_configuration_iterator (ParConfigLIter pc_iter)
  set modelPCIter
• ParConfigLIter parallel_configuration_iterator () const
  return modelPCIter
• void auto_graphics (const bool flag)
  set modelAutoGraphicsFlag to activate posting of graphics data within evaluate/synchronize functions (automatic
  graphics posting in the model as opposed to graphics posting at the strategy level).
• bool is_null () const
  function to check modelRep (does this envelope contain a letter)
• Model * model_rep () const
  returns modelRep for access to derived class member functions that are not mapped to the top Model level

Static Public Member Functions
• static void active_variables (const RealVector &config_vars, Model &model)
  set the specified configuration to the Model’s inactive vars, converting from real to integer or through index to string
  value as needed
• static void inactive_variables (const RealVector &config_vars, Model &model)
  set the specified configuration to the Model’s inactive vars, converting from real to integer or through index to string
  value as needed
• static void inactive_variables (const RealVector &config_vars, Model &model, Variables &updated_vars)

Protected Member Functions
• Model (BaseConstructor, ProblemDescDB &problem_db)
  constructor initializing the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)
• Model (LightWtBaseConstructor, ProblemDescDB &problem_db, ParallelLibrary &parallel_lib, const Shared-VariablesData &svd, const SharedResponseData &srd, const ActiveSet &set, short output_level)
  constructor initializing base class for derived model class instances constructed on the fly
• Model (LightWtBaseConstructor, ProblemDescDB &problem_db, ParallelLibrary &parallel_lib)
  constructor initializing base class for recast model instances
• virtual void derived_evaluate (const ActiveSet &set)
  portion of evaluate() specific to derived model classes
• virtual void derived_evaluate_nowait (const ActiveSet &set)
  portion of evaluate_nowait() specific to derived model classes
• virtual const IntResponseMap & derived_synchronize ()
  portion of synchronize() specific to derived model classes
• virtual const IntResponseMap & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to derived model classes
CHAPTER 13. CLASS DOCUMENTATION

- virtual void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  portion of init_communicators() specific to derived model classes
- virtual void derived_init_serial ()
  portion of init_serial() specific to derived model classes
- virtual void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  portion of set_communicators() specific to derived model classes
- virtual void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  portion of free_communicators() specific to derived model classes
- void set_ie_asynchronous_mode (int max_eval_concurrency)
  default logic for defining asynchEvalFlag and evaluationCapacity based on ie_pl settings
- void string_variable_max (const StringSetArray &ssa, size_t offset, Variables &vars)
  set the current value of each string variable offset + i to the longest string value found in the admissible string set ssa[i]
- void string_variable_max (const StringRealMapArray &srma, size_t offset, Variables &vars)
  set the current value of each string variable offset + i to the longest string value found in the admissible string map srma[i]
- SizetMultiArrayConstView initialize_x0_bounds (const SizetArray &original_dvv, bool &active_derivs, bool &inactive_derivs, RealVector &x0, RealVector &fd_lb, RealVector &fd_ub) const
  Initialize data needed for computing finite differences (active/inactive, center point, and bounds)
- Real forward_grad_step (size_t num_deriv_vars, size_t xj_index, Real x0_j, Real lb_j, Real ub_j)
  Compute the forward step for a finite difference gradient; updates shortStep.
- template<typename MetaType>
  void rekey_response_map (MetaType &meta_object, const IntResponseMap &resp_map, IntIntMap &id_map, IntResponseMap &resp_map_rekey, bool deep_copy.resp=false)
  rekey returned jobs matched in id_map into resp_map_rekey; unmatched jobs are cached within the meta_object
- template<typename MetaType >
  void rekey_synch (MetaType &meta_object, bool block, IntIntMap &id_map, IntResponseMap &resp_map_rekey, bool deep_copy.resp=false)
  synchronize via meta_object and rekey returned jobs matched in id_map into resp_map_rekey; unmatched jobs are cached within the meta_object

Protected Attributes

- Variables currentVariables
  the set of current variables used by the model for performing function evaluations
- size_t numDerivVars
  the number of active continuous variables used in computing most response derivatives (i.e., in places such as quasi-Hessians and response corrections where only the active continuous variables are supported)
- Response currentResponse
  the set of current responses that holds the results of model function evaluations
- size_t numFns
  the number of functions in currentResponse
- Constraints userDefinedConstraints
Explicit constraints on variables are maintained in the Constraints class hierarchy. Currently, this includes linear constraints and bounds, but could be extended in the future to include other explicit constraints which (1) have their form specified by the user, and (2) are not catalogued in Response since their form and coefficients are published to an iterator at startup.

- **String modelId**
  - model identifier string from the input file
- **String modelType**
  - type of model: simulation, nested, or surrogate
- **String surrogateType**
  - type of surrogate model: local, multipoint, global, or hierarchical
- **String gradientType**
  - type of gradient data: analytic, numerical, mixed, or none
- **String methodSource**
  - source of numerical gradient routine: dakota or vendor
- **String intervalType**
  - type of numerical gradient interval: central or forward
- **String hessianType**
  - type of Hessian data: analytic, numerical, quasi, mixed, or none
- **RealVector fdGradStepSize**
  - relative finite difference step size for numerical gradients
- **String fdGradStepType**
  - type of finite difference step to use for numerical gradient: relative - step length is relative to $x$ absolute - step length is what is specified bounds - step length is relative to range of $x$
- **RealVector fdHessByGradStepSize**
  - relative finite difference step size for numerical Hessians estimated using first-order differences of gradients
- **RealVector fdHessByFnStepSize**
  - relative finite difference step size for numerical Hessians estimated using second-order differences of function values
- **String fdHessStepType**
  - type of finite difference step to use for numerical Hessian: relative - step length is relative to $x$ absolute - step length is what is specified bounds - step length is relative to range of $x$
- **bool ignoreBounds**
  - option to ignore bounds when computing finite diffs
- **bool centralHess**
  - option to use old 2nd-order finite diffs for Hessians
- **bool warmStartFlag**
  - if in warm-start mode, don’t reset accumulated data (e.g., quasiHessians)
- **bool supportsEstimDerivs**
  - whether model should perform or forward derivative estimation
- **String quasiHessType**
  - quasi-Hessian type: bfgs, damped_bfgs, sr1
- **IntSet gradIdAnalytic**
  - analytic id’s for mixed gradients
- **IntSet gradIdNumerical**
  - numerical id’s for mixed gradients
• IntSet hessIdAnalytic
  analytic id’s for mixed Hessians
• IntSet hessIdNumerical
  numerical id’s for mixed Hessians
• IntSet hessIdQuasi
  quasi id’s for mixed Hessians
• IntArray messageLengths
  length of packed MPI buffers containing vars, vars/set, response, and PRPair
• ProblemDescDB & probDescDB
  class member reference to the problem description database
• ParallelLibrary & parallelLib
  class member reference to the parallel library
• ParConfigLIter modelPCIter
  the ParallelConfiguration node used by this Model instance
• short componentParallelMode
  the component parallelism mode: 0 (none), 1 (INTERFACE/OPTIONAL_INTERFACE/SURROGATE_MODEL), or 2 (SUB_MODEL/TRUTH_MODEL)
• bool asynchEvalFlag
  flags asynch evaluations (local or distributed)
• int evaluationCapacity
  capacity for concurrent evaluations supported by the Model
• short outputLevel
  output verbosity level: {SILENT, QUIET, NORMAL, VERbose, DEBUG}, OUTPUT
• IntSetArray discreteDesignSetIntValues
  array of IntSet’s, each containing the set of allowable integer values corresponding to a discrete design integer set variable
• StringSetArray discreteDesignSetStringValues
  array of IntSet’s, each containing the set of allowable integer values corresponding to a discrete design string set variable
• RealSetArray discreteDesignSetRealValues
  array of RealSet’s, each containing the set of allowable real values corresponding to a discrete design real set variable
• IntSetArray discreteStateSetIntValues
  array of IntSet’s, each containing the set of allowable integer values corresponding to a discrete state integer set variable
• StringSetArray discreteStateSetStringValues
  array of IntSet’s, each containing the set of allowable integer values corresponding to a discrete state string set variable
• RealSetArray discreteStateSetRealValues
  array of RealSet’s, each containing the set of allowable real values corresponding to a discrete state real set variable
• Pecos::AleatoryDistParams aleatDistParams
  container for aleatory random variable distribution parameters
• Pecos::EpistemicDistParams epistDistParams
container for epistemic random variable distribution parameters

- `BoolDeque primaryRespFnSense`
  array of flags (one per primary function) for switching the sense to maximize the primary function (default is minimize)

- `RealVector primaryRespFnWts`
  primary response function weightings (either weights for multiobjective optimization or weighted least squares)

- `bool hierarchicalTagging`
  whether to perform hierarchical evalID tagging of params/results

- `ScalingOptions scalingOpts`
  user-provided scaling data from the problem DB, possibly modified by Recasting

- `String evalTagPrefix`
  cached evalTag Prefix from parents to use at evaluate time

Private Member Functions

- `Model * get_model (ProblemDescDB &problem_db)`
  Used by the envelope to instantiate the correct letter class.

- `int estimate_derivatives (const ShortArray &map_asv, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, const bool asynch_flag)`
  evaluate numerical gradients using finite differences. This routine is selected with "method_source dakota" (the default method_source) in the numerical gradient specification.

- `void synchronize_derivatives (const Variables &vars, const IntResponseMap &fd_responses, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set)`
  combine results from an array of finite difference response objects (fd_grad_responses) into a single response (new_response)

- `void update_response (const Variables &vars, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, Response &initial_map_response, const RealMatrix &new_fn_grads, const RealSymMatrixArray &new_fnhessians)`
  overlay results to update a response object

- `void update_quasi_hessians (const Variables &vars, Response &new_response, const ActiveSet &original_set)`
  perform quasi-Newton Hessian updates

- `Real finite_difference_lower_bound (UShortMultiArrayConstView cv_types, const RealVector &global_c_l_bnds, size_t cv_index) const`
  return the lower bound for a finite difference offset, drawn from global or distribution bounds

- `Real finite_difference_upper_bound (UShortMultiArrayConstView cv_types, const RealVector &global_c_u_bnds, size_t cv_index) const`
  return the upper bound for a finite difference offset, drawn from global or distribution bounds

- `bool manage_asv (const ActiveSet &original_set, ShortArray &map_asv_out, ShortArray &fd_grad_asv_out, ShortArray &fd_hess_asv_out, ShortArray &quasi_hess_asv_out)`
  Coordinates usage of estimate_derivatives() calls based on asv_in.

- `Real initialize_h (Real x_j, Real lb_j, Real ub_j, Real step_size, String step_type) const`
  function to determine initial finite difference h (before step length adjustment) based on type of step desired
• Real FDstep1 (Real x0_j, Real lb_j, Real ub_j, Real h_mag)
  function returning finite-difference step size (affected by bounds)
• Real FDstep2 (Real x0_j, Real lb_j, Real ub_j, Real h)
  function returning second central-difference step size (affected by bounds)

Private Attributes
• int modelEvalCntr
  evaluation counter for top-level evaluate() and evaluate_nowait() calls. Differs from lower level counters in case of numerical derivative estimation (several lower level evaluations are assimilated into a single higher level evaluation)
• bool estDerivsFlag
  flags presence of estimated derivatives within a set of calls to evaluate_nowait()
• bool shortStep
  flags finite-difference step size adjusted by bounds
• std::map< SizetIntPair, ParConfigIter > modelPCIterMap
  map<> used for tracking modelPCIter instances using depth of parallelism level and max evaluation concurrency as the lookup keys
• bool initCommsBcastFlag
  flag for determining need to bcast the max concurrency from init_communicators(); set from IteratorScheduler::init_iterator()
• bool modelAutoGraphicsFlag
  flag for posting of graphics data within evaluate() (automatic graphics posting in the model as opposed to graphics posting at the strategy level)
• IntVariablesMap varsMap
  history of vars populated in evaluate_nowait() and used in synchronize().
• std::list< ShortArray > asvList
  if estimate_derivatives() is used, transfers ASVs from evaluate_nowait() to synchronize()
• std::list< ActiveSet > setList
  if estimate_derivatives() is used, transfers ActiveSets from evaluate_nowait() to synchronize()
• BoolList initialMapList
  transfers initial_map flag values from estimate_derivatives() to synchronize_derivatives()
• BoolList dbCaptureList
  transfers db_capture flag values from estimate_derivatives() to synchronize_derivatives()
• ResponseList dbResponseList
  transfers database captures from estimate_derivatives() to synchronize_derivatives()
• RealList deltaList
  transfers deltas from estimate_derivatives() to synchronize_derivatives()
• IntIntMap numFDEvalsMap
  tracks the number of evaluations used within estimate_derivatives(). Used in synchronize() as a key for combining finite difference responses into numerical gradients.
• IntIntMap rawEvalIdMap
  maps from the raw evaluation ids returned by derived_synchronize() and derived_synchronize_nowait() to the corresponding modelEvalCntr id. Used for rekeying responseMap.
13.83. Model Class Reference

- **RealVectorArray** `xPrev`  
  previous parameter vectors used in computing $s$ for quasi-Newton updates
- **RealMatrix** `fnGradsPrev`  
  previous gradient vectors used in computing $y$ for quasi-Newton updates
- **RealSymMatrixArray** `quasiHessians`  
  quasi-Newton Hessian approximations
- **SizetArray** `numQuasiUpdates`  
  number of quasi-Newton Hessian updates applied
- **IntResponseMap** `responseMap`  
  used to return a map of responses for asynchronous evaluations in final concatenated form. The similar map in Interface contains raw responses.
- **IntResponseMap** `cachedResponseMap`  
  caching of responses returned by derived `synchronize{,nowait}()` but not matched within current rawEvalIdMap
- **IntResponseMap** `graphicsRespMap`  
  used to cache the data returned from derived `synchronize_nowait()` prior to sequential input into the graphics
- **IntSetArray** `activeDiscSetIntValues`  
  aggregation of the admissible value sets for all active discrete set integer variables
- **StringSetArray** `activeDiscSetStringValues`  
  aggregation of the admissible value sets for all active discrete set string variables
- **RealSetArray** `activeDiscSetRealValues`  
  aggregation of the admissible value sets for all active discrete set real variables
- **BitArray** `discreteIntSets`  
  key for identifying discrete integer set variables within the active discrete integer variables
- **ModelList** `modelList`  
  used to collect sub-models for `subordinate_models()`
- **BoolDeque** `recastFlags`  
  a key indicating which models within a model recursion involve recasting
- **Model** `*modelRep`  
  pointer to the letter (initialized only for the envelope)
- **int** `referenceCount`  
  number of objects sharing modelRep

**Friends**

- **bool** `operator==` (const Model &m1, const Model &m2)  
  equality operator (detect same letter instance)
- **bool** `operator!=` (const Model &m1, const Model &m2)  
  inequality operator (detect different letter instances)

### 13.83.1 Detailed Description

Base class for the model class hierarchy.

The **Model** class is the base class for one of the primary class hierarchies in DAKOTA. The model hierarchy contains a set of variables, an interface, and a set of responses, and an iterater operates on the model to map the variables into responses using the interface. For memory efficiency and enhanced polymorphism, the model hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Model) serves as the envelope and one of the derived classes (selected in `Model::get_model()`) serves as the letter.
13.83.2 Constructor & Destructor Documentation

Model ( )

default constructor

The default constructor is used in vector<Model> instantiations and for initialization of Model objects contained in Iterator and derived Strategy classes. modelRep is NULL in this case (a populated problem_db is needed to build a meaningful Model object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

Model ( ProblemDescDB & problem_db )

standard constructor for envelope

Used in model instantiations within strategy constructors. Envelope constructor only needs to extract enough data to properly execute get_model, since Model(BaseConstructor, problem_db) builds the actual base class data for the derived models.

References Dakota::abort_handler(), Model::get_model(), and Model::modelRep.

Model ( const Model & model )

copy constructor

Copy constructor manages sharing of modelRep and incrementing of referenceCount.

References Model::modelRep, and Model::referenceCount.

~Model ( ) [virtual]

destructor

Destructor decrements referenceCount and only deletes modelRep when referenceCount reaches zero.

References Model::modelRep, and Model::referenceCount.

Model ( BaseConstructor, ProblemDescDB & problem_db ) [protected]

constructor initializing the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor builds the base class data for all inherited models. get_model() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_model() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Model).

References Dakota::abort_handler(), Model::currentResponse, Model::fdGradStepSize, Model::fdHessByFnStepSize, Model::fdHessByGradStepSize, ProblemDescDB::get_sa(), Model::gradIdNumerical, Model::gradientType, Model::hessianType, Model::hessIdNumerical, Response::num_functions(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Model::primaryRespFnSense, Dakota::strbegins(), Dakota::strtolower(), and Model::userDefinedConstraints.

Model ( LightWtBaseConstructor, ProblemDescDB & problem_db, ParallelLibrary & parallel_lib ) [protected]

constructor initializing base class for recast model instances

This constructor also builds the base class data for inherited models. However, it is used for recast models which are instantiated on the fly. Therefore it only initializes a small subset of attributes.
13.83.3 Member Function Documentation

Model operator=( const Model & model )

assignment operator

Assignment operator decrements referenceCount for old modelRep, assigns new modelRep, and increments
referenceCount for new modelRep.

References Model::modelRep, and Model::referenceCount.

Iterator & subordinate_iterator () [virtual]

return the sub-iterator in nested and surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this
operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and NestedModel.

References Dakota::dummy_iterator, Model::modelRep, and Model::subordinate_iterator().

Referenced by NonDExpansion::annotated_refinement_results(), NonDExpansion::annotated_results(), Non-
DPolynomialChaos::append_expansion(), NonDExpansion::compute_expansion(), SurrBasedGlobalMinimizer::core-
_run(), NonDExpansion::finalize_sets(), NonDGlobalReliability::get_best_sample(), NonDPolynomialChaos::increment-
_grid_from_order(), NonDMultilevelPolynomialChaos::increment_sample_sequence(), NonDExpansion::increment-
_sets(), NonDMultilevelStochCollocation::increment_specification_sequence(), NonDMultilevelPolynomialChaos-
::increment_specification_sequence(), NonExpansion::initialize_expansion(), NonDExpansion::initialize_sets(),
NonDStochCollocation::initialize_u_space_model(), NonDExpansion::initialize_u_space_model(), NonDPolynomial-
Chaos::initialize_u_space_model(), NonDMultilevelPolynomialChaos::multilevel_regression(), NonDLocalInterval-
::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDExpansion::refine_expansion(), S-
OLBase::SOLBase(), Model::subordinate_iterator(), RecastModel::subordinate_iterator(), NonDStochCollocation-
::update_expansion(), and NonDQUESOBayesCalibration::update_model().

Model & subordinate_model() [virtual]

return a single sub-model defined from subModel in nested and recast models and truth_model() in surrogate
models; used for a directed dive through model recursions that may bypass some components.

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this
operation must be performed on the original envelope object.

Reimplemented in RecastModel, NestedModel, and SurrogateModel.

References Dakota::dummy_model, Model::modelRep, and Model::subordinate_model().

Referenced by NonDGlobalReliability::expected_feasibility(), NonDGlobalReliability::expected_improvement(),
SurrogateModel::force_rebuild(), AdaptedBasisModel::get_sub_model(), NonDExpansion::initialize_expansion(),
Minimizer::initialize_run(), NonDExpansion::initialize_u_space_model(), NonDGlobalReliability::optimize_gaussian-
_process(), Minimizer::original_model(), Optimizer::primary_response_reducer(), LeastSq::primary_response_weighter(), Model-
::subordinate_model(), DataFitSurrModel::update_global(), and LeastSq::weight_model().

Model & surrogate_model() [virtual]

return the active approximation sub-model in surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this
operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and HierarchSurrModel.

References Dakota::dummy_model, Model::modelRep, and Model::surrogate_model().
Referenced by NonDAdaptiveSampling::calc_score_delta_y(), NonDAdaptiveSampling::calc_score_topo_alm_hybrid(), NonDAdaptiveSampling::calc_score_topo_avg_persistence(), NonDAdaptiveSampling::calc_score_topo_bottleneck(), NonBayesCalibration::construct_mcmc_model(), NonMultilevelSampling::control_variate_mc(), SurrBasedGlobalMinimizer::core_run(), SurrBasedLocalMinimizer::find_approx_response(), NonMultilevelSampling::lf_increment(), NonMultilevelSampling::multilevel_control_variate_mc_Qcorr(), NonMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonMultilevelSampling::multilevel_mc_Qsum(), NonMultilevelSampling::multilevel_mc_Ysum(), NonDAdaptiveSampling::output_round_data(), Model::surrogate_model(), and RecastModel::surrogate_model().

**Model & truth_model()** [virtual]

return the active truth sub-model in surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and HierarchSurrModel.

References Dakota::dummy_model, Model::modelRep, and Model::truth_model().

Referenced by NonMultilevelSampling::compute_error_estimates(), NonMultilevelSampling::control_variate_mc(), SurrBasedGlobalMinimizer::core_run(), SurrBasedLocalMinimizer::find_truth_response(), SurrogateModel::force_rebuild(), SurrBasedMinimizer::initialize_graphics(), NonDExpansion::multifidelity_expansion(), NonMultilevelSampling::multilevel_control_variate_mc_Qcorr(), NonMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonMultilevelSampling::multilevel_mc_Qsum(), NonMultilevelSampling::multilevel_mc_Ysum(), NonBayes Calibration::NonBayesCalibration(), NonDLocalReliability::NonDLocalReliability(), SurrBasedMinimizer::print_results(), NonMultilevelSampling::print_results(), NonMultilevelSampling::shared_increment(), SurrogateModel::subordinate_model(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), Model::truth_model(), and RecastModel::truth_model().

```cpp
void update_from_subordinate_model ( size_t depth = std::numeric_limits<size_t>::max() ) [virtual]
```

propagate vars/labels/bounds/targets from the bottom up

used only for instantiate-on-the-fly model recursions (all RecastModel instantiations and alternate DataFitSurrModel instantiations). Simulation, Hierarchical, and Nested Models do not redefine the function since they do not support instantiate-on-the-fly. This means that the recursion will stop as soon as it encounters a Model that was instantiated normally, which is appropriate since ProblemDescDB-constructed Models use top-down information flow and do not require bottom-up updating.

Reimplemented in RecastModel, HierarchSurrModel, and DataFitSurrModel.

References Model::modelRep, and Model::update_from_subordinate_model().

Referenced by NonDGlobalInterval::core_run(), NonDLocalInterval::core_run(), NonDLocalReliability::initialize_class_data(), NonDExpansion::initialize_expansion(), LeastSq::initialize_run(), Optimizer::initialize_run(), EffGlobalMinimizer::minimize_surrogates_on_model(), NonMultilevelPolynomialChaos::multilevel_regression(), NonDGlobalReliability::optimize_gaussian_process(), DataFitSurrModel::update_from_subordinate_model(), HierarchSurrModel::update_from_subordinate_model(), Model::update_from_subordinate_model(), and RecastModel::update_from_subordinate_model().

**Interface & derived_interface()** [virtual]

return the interface employed by the derived model class, if present: SimulationModel::userDefinedInterface, DataFitSurrModel::approxInterface, or NestedModel::optionalInterface

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.
Reimplemented in `RecastModel`, `DataFitSurrModel`, `NestedModel`, and `SimulationModel`. References `Model::derived_interface()`, `Dakota::dummy_interface`, and `Model::modelRep`. Referenced by `SurrBasedGlobalMinimizer::core_run()`, `Model::derived_interface()`, and `RecastModel::derived_interface()`.

```cpp
size_t solution_levels ( bool lwr_bnd = true ) const [virtual]
```

number of discrete levels within solution control (`SimulationModel`) return the number of levels within a solution / discretization hierarchy. Reimplemented in `SimulationModel`. References `Model::modelRep`, and `Model::solution_levels()`. Referenced by `NonDMultilevelSampling::compute_error_estimates()`, `NonDExpansion::multifidelity_expansion()`, `NonDMultilevelSampling::multilevel_control_variate_mc_Qcorr()`, `NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr()`, `NonDMultilevelSampling::multilevel_mc_Qsum()`, `NonDMultilevelSampling::multilevel_mc_Ysum()`, `Model::solution_levels()`, and `RecastModel::solution_levels()`.

```cpp
void solution_level_index ( size_t index ) [virtual]
```

activate a particular level within the solution level control and return the cost estimate (`SimulationModel`) activate a particular level within a solution / discretization hierarchy. Reimplemented in `RecastModel`, and `SimulationModel`. References `Dakota::abort_handler()`, `Model::modelRep`, and `Model::solution_level_index()`. Referenced by `HierarchSurrModel::build_approximation()`, `HierarchSurrModel::derived_evaluate()`, `HierarchSurrModel::derived_evaluate_nowait()`, `Model::solution_level_index()`, and `RecastModel::solution_level_index()`.

```cpp
short local_eval_synchronization ( ) [virtual]
```

return derived model synchronization setting

```
SimulationModels and HierarchSurrModels redefine this virtual function.
A default value of "synchronous" prevents asynch local operations for:

- NestedModels: a subIterator can support message passing parallelism, but not asynch local.
- DataFitSurrModels: while asynch evals on approximations will work due to some added bookkeeping, avoiding them is preferable.
```

Reimplemented in `RecastModel`, `NestedModel`, and `SimulationModel`. References `Model::local_eval_synchronization()`, and `Model::modelRep`. Referenced by `Model::init_serial()`, `RecastModel::local_eval_synchronization()`, `Model::local_eval_synchronization()`, and `Model::set_ie_asynchronous_mode()`.

```cpp
int local_eval_concurrency ( ) [virtual]
```

return derived model asynchronous evaluation concurrency

```
SimulationModels and HierarchSurrModels redefine this virtual function.
Reimplemented in `RecastModel`, `NestedModel`, and `SimulationModel`. References `Model::local_eval_concurrency()`, and `Model::modelRep`. Referenced by `RecastModel::local_eval_concurrency()`, `Model::local_eval_concurrency()`, and `Model::set_ie_asynchronous_mode()`.
const String & interface_id() const [virtual]

return the interface identifier

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, NestedModel, and SimulationModel.

References Dakota::dummy_interface, Interface::interface_id(), Model::interface_id(), and Model::modelRep.

Referenced by NonDQUESOBayesCalibration::aggregate_acceptance_chain(), NonDDREAMBayesCalibration::archive_acceptance_chain(), DataFitSurrModel::build_global(), DataFitSurrModel::DataFitSurrModel(), Model::db_lookup(), Model::estimate_message_lengths(), Model::evaluate(), NonDMultilevelSampling::export_all_samples(), NonDBayesCalibration::export_chain(), NonDBayesCalibration::export_discrepancy(), SurrBasedLocalMinimizer::find_approx_response(), SurrBasedLocalMinimizer::find_truth_response(), DataFitSurrModel::import_points(), RecastModel::interface_id(), Model::interface_id(), Minimizer::local_recast_retrieve(), Analyzer::pre_output(), SurrBasedMinimizer::print_results(), LeastSq::print_results(), Optimizer::print_results(), SeqHybridMetaIterator::run_sequential(), DiscrepancyCorrection::search_db(), Model::synchronize(), Model::synchronize_nowait(), Analyzer::update_best(), ConcurrentMetaIterator::update_local_results(), SeqHybridMetaIterator::update_local_results(), and NonDLocalReliability::update_mpp_search_data().

bool evaluation_cache ( bool recurse_flag = true ) const [virtual]

Indicates the usage of an evaluation cache by the Model.

Only Models including ApplicationInterfaces support an evaluation cache: surrogate, nested, and recast mappings are not stored in the cache. Possible exceptions: HierarchSurrModel, NestedModel::optionalInterface.

Reimplemented in RecastModel, DataFitSurrModel, HierarchSurrModel, and SimulationModel.

References Model::evaluation_cache(), and Model::modelRep.

Referenced by DataFitSurrModel::DataFitSurrModel(), DataFitSurrModel::evaluation_cache(), RecastModel::evaluation_cache(), Model::evaluation_cache(), DataFitSurrModel::import_points(), and Analyzer::read_variables_responses().

bool restart_file ( bool recurse_flag = true ) const [virtual]

Indicates the usage of a restart file by the Model.

Only Models including ApplicationInterfaces interact with the restart file: surrogate, nested, and recast mappings are not stored in restart. Possible exceptions: DataFitSurrModel::import_points(), NestedModel::optionalInterface.

Reimplemented in RecastModel, DataFitSurrModel, HierarchSurrModel, and SimulationModel.

References Model::modelRep, and Model::restart_file().

Referenced by DataFitSurrModel::import_points(), Analyzer::read_variables_responses(), DataFitSurrModel::restart_file(), RecastModel::restart_file(), and Model::restart_file().

void eval_tag_prefix ( const String & eval_id_str ) [virtual]

set the hierarchical eval ID tag prefix

Derived classes containing additional models or interfaces should implement this function to pass along to their sub Models/Interfaces.

Reimplemented in RecastModel, and SimulationModel.

References Model::eval_tag_prefix(), Model::evalTagPrefix, and Model::modelRep.

Referenced by HierarchSurrModel::build_approximation(), HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), DataFitSurrModel::derived_evaluate_nowait(), Iterator::eval_tag_prefix(), RecastModel::eval_tag_prefix(), and Model::eval_tag_prefix().
13.83. MODEL CLASS REFERENCE

ModelList & subordinate_models ( bool recurse_flag = true )

return the sub-models in nested and surrogate models

since modelList is built with list insertions (using envelope copies), these models may not be used for model::assign_rep() since this operation must be performed on the original envelope object. They may, however, be used for letter-based operations (including assign_rep() on letter contents such as an interface).

References Model::derived_subordinate_models(), Model::modelList, Model::modelRep, and Model::subordinate_models().

Referenced by Model::manage_data_recastings(), NonDExpansion::multifidelity_expansion(), NonDMultilevel-PolynomialChaos::multilevel_regression(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDMultilevelSampling::NonDMultilevelSampling(), SOLBase::SOLBase(), and Model::subordinate_models().

void init_communicators ( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag = true )

allocate communicator partitions for a model and store configuration in modelPCIterMap

The init_communicators() and derived_init_communicators() functions are structured to avoid performing the messageLengths estimation more than once. init_communicators() (not virtual) performs the estimation and then forwards the results to derived_init_communicators (virtual) which uses the data in different contexts.

References ParallelLibrary::bcast(), Model::derived_init_communicators(), Model::estimate_message_lengths(), ParallelLibrary::increment_parallel_configuration(), Model::init_communicators(), Model::initCommsBcastFlag, Model::messageLengths, Model::modelPCIter, Model::modelPCIterMap, Model::modelRep, ParallelLibrary::parallel_configuration_iterator(), ParallelLibrary::parallel_level_index(), and Model::parallelLib.

Referenced by SurrBasedMinimizer::derived_init_communicators(), NonDGlobalReliability::derived_init_communicators(), NonDLocalInterval::derived_init_communicators(), NonDExpansion::derived_init_communicators(), NonDGlobal-Interval::derived_init_communicators(), NonDExpand::derived_init_communicators(), NonDAdaptImpSampling::derived_init_communicators(), NonDGPImpSampling::derived_init_communicators(), NonDAdaptiveSampling::derived_init_communicators(), NonDLocalReliability-::derived_init_communicators(), AdaptedBasisModel::derived_init_communicators(), NonDPolynomialChaos::derived_init_communicators(), NonDBayesCalibration::derived_init_communicators(), ActiveSubspaceModel::derived_init_communicators(), HierarchSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_communicators(), RecastModel::derived_init_communicators(), Iterator::derived_init_communicators(), DataFitSurrModel::derived_set_communicators(), Model::init_communicators(), and Model::serve_init_communicators().

void init_serial ( )

for cases where init_communicators() will not be called, modify some default settings to behave properly in serial.

The init_serial() and derived_init_serial() functions are structured to separate base class (common) operations from derived class (specialized) operations.

References Model::asynchEvalFlag, Model::derived_init_serial(), Model::init_serial(), Model::local_eval_synchronization(), and Model::modelRep.

Referenced by NestedModel::derived_init_serial(), HierarchSurrModel::derived_init_serial(), DataFitSurrModel::derived_init_serial(), RecastModel::derived_init_serial(), and Model::init_serial().

void estimate_message_lengths ( )

estimate messageLengths for a model

This functionality has been pulled out of init_communicators() and defined separately so that it may be used in those cases when messageLengths is needed but model.init_communicators() is not called, e.g., for the master processor in the self-scheduling of a concurrent iterator strategy.

References Response::active_set_derivative_vector(), Model::aleatDistParams, Response::copy(), Variables::copy(), Model::currentResponse, Model::currentVariables, Variables::cv(), Model::discreteDesignSetStringValues,
CHAPTER 13. CLASS DOCUMENTATION

Model::discreteStateSetStringValues, Model::epistDistParams, Model::estimate_message_lengths(), Variables::icv(), Model::interface_id(), Model::messageLengths, Model::modelRep, ParallelLibrary::mpirun_flag(), Model::num-Fns, Model::parallelLib, MPIPackBuffer::reset(), MPIPackBuffer::size(), and Model::string_variable_max().

Referenced by Model::estimate_message_lengths(), Model::init_communicators(), ProbabilityTransformModel::initialize_mapping(), RandomFieldModel::initialize_mapping(), ConcurrentMetaIterator::pre_run(), Iterator::resize_communicators(), AdaptedBasisModel::serve_init_mapping(), and ActiveSubspaceModel::serve_init_mapping().

bool manage_data_recastings ( )

initialize modelList and recastFlags for data import/export

Constructor helper to manage model recastings for data import/export.

References Model::manage_data_recastings(), Model::modelRep, Model::recastFlags, and Model::subordinate_models().

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::manage_data_recastings(), and Analyzer::read_variables_responses().

void assign_rep ( Model * model_rep, bool ref_count_incr = true )

replaces existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old modelRep and assigns the new modelRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.
- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_model(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), Model::model_rep(), Model::modelRep, and Model::referenceCount.

Referenced by ActiveSubspaceModel::build_surrogate(), NonDBayesCalibration::calibrate_to_hifi(), ActiveSubspaceModel::computeCrossValidationMetric(), NonDBayesCalibration::construct_map_optimizer(), NonDBayesCalibration::construct_mcmc_model(), EffGlobalMinimizer::EffGlobalMinimizer(), ActiveSubspaceModel::get_sub_model(), SurrBasedLocalMinimizer::initialize_sub_model(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGlobalInterval::NonDGlobalInterval(), NonDGPImpSampling::NonDGPImpSampling(), NonDAdaptiveModel::nonDAdaptiveModel(), NonDActiveSubspace::nonDActiveSubspace(), NonDConcurrentMeta::nonDConcurrentMeta(), NonDDeconvolution::nonDDeconvolution(), NonDEffGlobalMinimizer::nonDEffGlobalMinimizer(), NonDGlobalInterval::nonDGlobalInterval(), NonDGlobalReliability::nonDGlobalReliability(), NonDGPImpSampling::nonDGPImpSampling(), NonDLocalInterval::nonDLocalInterval(), NonDLocalReliability::nonDLocalReliability(), NonDMultilevelPolynomialChaos::nonDMultilevelPolynomialChaos(), NonDMultilevelStochCollocation::nonDMultilevelStochCollocation(), NonDPolynomialChaos::nonDPolynomialChaos(), NonDStochCollocation::nonDStochCollocation(), Optimizer::reduce_model(), NonDPolynomialChaos::resize(), Minimizer::scale_model(), NonD::transform_model(), and LeastSq::weight_model().

int derivative_concurrency ( ) const

return the gradient concurrency for use in parallel configuration logic

This function assumes derivatives with respect to the active continuous variables. Therefore, concurrency with respect to the inactive continuous variables is not captured.
References Dakota::contains(), Model::derivative_concurrency(), Model::gradIdAnalytic, Model::gradientType, Model::hessianType, Model::hessIdNumerical, Model::intervalType, Model::methodSource, Model::modelRep, and Model::numDerivVars.

Referenced by ActiveSubspaceModel::ActiveSubspaceModel(), Model::derivative_concurrency(), HierarchSurrModel::derived_free_communicators(), HierarchSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_communicators(), HierarchSurrModel::derived_set_communicators(), DataFitSurrModel::estimate_partition_bounds(), NonDExpansion::initialize_u_space_model(), Analyzer::num_samples(), HierarchSurrModel::serve_run(), and Iterator::update_from_model().

void active_variables ( const RealVector & config_vars, Model & model ) [static]
set the specified configuration to the Model’s inactive vars, converting from real to integer or through index to string value as needed
config_vars consists of [continuous, integer, string, real].
References Model::continuous_variables(), Model::current_variables(), Model::cv(), Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_string_values(), Variables::discrete_string_variable(), Model::div(), Model::drv(), Model::dsv(), Dakota::iround(), and Dakota::set_index_to_value().

void inactive_variables ( const RealVector & config_vars, Model & model ) [static]
set the specified configuration to the Model’s inactive vars, converting from real to integer or through index to string value as needed
config_vars consists of [continuous, integer, string, real].
References Model::current_variables().
Referenced by NonDBayesCalibration::build_model_discrepancy(), NonDBayesCalibration::calibrate_to_hifi(), DataTransformModel::derived_evaluate(), DataTransformModel::derived_evaluate_nowait(), and NonDBayesCalibration::export_discrepancy().

void inactive_variables ( const RealVector & config_vars, Model & model, Variables & vars ) [static]
config_vars consists of [continuous, integer, string, real].
References Model::current_variables(), Model::discrete_set_string_values(), Model::icv(), Model::idiv(), Model::idrv(), Model::idsv(), Variables::inactive_continuous_variables(), Variables::inactive_discrete_int_variables(), Variables::inactive_discrete_real_variables(), Variables::inactive_discrete_string_variable(), Dakota::iround(), Dakota::set_index_to_value(), and Variables::view().

Model * get_model ( ProblemDescDB & problem_db ) [private]
Used by the envelope to instantiate the correct letter class.
Used only by the envelope constructor to initialize modelRep to the appropriate derived type, as given by the modelType attribute.
References ProblemDescDB::get_string(), Model::model_type(), and Model::modelType.
Referenced by Model::Model().

int estimate_derivatives ( const ShortArray & map_asv, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set, const bool asynch_flag ) [private]
evaluate numerical gradients using finite differences. This routine is selected with “method_source dakota” (the default method_source) in the numerical gradient specification.
Estimate derivatives by computing finite difference gradients, finite difference Hessians, and/or quasi-Newton Hessians. The total number of finite difference evaluations is returned for use by `synchronize()` to track response arrays, and it could be used to improve management of max_function_evaluations within the iterators.

References Variables::all_continuous_variables(), Model::centralHess, Variables::continuous_variables(), Model::currentResponse, Model::currentVariables, Model::db_lookup(), Model::dbCaptureList, Model::dbResponseList, Model::deltaList, ActiveSet::derivative_vector(), Model::derived_evaluate(), Model::derived_evaluate_nowait(), Model::fdHessByFnStepSize, Model::fdHessByGradStepSize, Model::fdHessStepType, Model::FDstep1(), Model::FDstep2(), Dakota::find_index(), Model::forward_grad_step(), Response::function_gradients(), Response::function_values(), Model::ignoreBounds, Variables::inactive_continuous_variables(), Model::initialize_h(), Model::initialize_x0_bounds(), Model::initialMapList, Model::intervalType, Model::numFns, Model::outputLevel, ActiveSet::request_vector(), Response::shared_data(), Model::shortStep, and Model::update_response.

Referenced by Model::evaluate(), and Model::evaluate_nowait().

```cpp
void synchronize_derivatives (const Variables & vars, const IntResponseMap & fd_responses, Response & new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set) [private]
```

combine results from an array of finite difference response objects (fd_grad_responses) into a single response (new_response)

Merge an array of fd_responses into a single new_response. This function is used both by synchronous evaluate() for the case of asynchronous estimate_derivatives() and by synchronize() for the case where one or more evaluate_nowait() calls has employed asynchronous estimate_derivatives().

References Model::acv(), Variables::all_continuous_variable_ids(), Model::centralHess, Variables::continuous_variable_ids(), Model::currentResponse, Model::currentVariables, Model::cv(), Model::dbCaptureList, Model::dbResponseList, Model::deltaList, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradients(), Response::function_values(), Model::icv(), Variables::inactive_continuous_variable_ids(), Model::initialMapList, Model::intervalType, Model::numFns, Response::shared_data(), and Model::update_response.

Referenced by Model::evaluate(), and Model::synchronize().

```cpp
void update_response (const Variables & vars, Response & new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const RealMatrix & new_fn_grads, const RealSymMatrixArray & new_fn_hessians) [private]
```

overlay results to update a response object

Overlay the initial_map_response with numerically estimated new_fn_grads and new_fn_hessians to populate new_response as governed by asv vectors. Quasi-Newton secant Hessian updates are also performed here, since this is where the gradient data needed for the updates is first consolidated. Convenience function used by estimate_derivatives() for the synchronous case and by synchronize_derivatives() for the asynchronous case.

References Response::active_set_request_vector(), Variables::continuous_variable_ids(), Response::copy(), Model::currentResponse, Model::currentVariables, ActiveSet::derivative_vector(), Response::function_gradients(), Response::function_hessians(), Response::function_values(), Model::hessianType, Model::hessIdQuasi, Response::is_null(), Model::numFns, Model::outputLevel, Model::quasiHessians, ActiveSet::request_vector(), Response::reset_inactive(), Model::supportsEstimDerivs, Model::surrogate_response_mode(), and Model::update_quasi_hessians.

Referenced by Model::estimate_derivatives(), and Model::synchronize_derivatives().

```cpp
void update_quasi_hessians (const Variables & vars, Response & new_response, const ActiveSet & original_set) [private]
```

perform quasi-Newton Hessian updates
quasi-Newton updates are performed for approximating response function Hessians using BFGS or SR1 formulations. These Hessians are supported only for the active continuous variables, and a check is performed on the DVV prior to invoking the function.

References Dakota::contains(), Variables::continuous_variables(), Model::fnGradsPrev, Response::function_gradients(), Model::hessianType, Model::hessIdQuasi, Model::modelType, Model::numDerivVars, Model::numFns, Model::numQuasiUpdates, Model::outputLevel, Model::quasiHessians, Model::quasiHessType, ActiveSet::request_vector(), and Model::xPrev.

Referenced by Model::update_response().

```cpp
bool manage_asv ( const ActiveSet & original_set, ShortArray & map_asv_out, ShortArray & fd_grad_asv_out, ShortArray & fd_hess_asv_out, ShortArray & quasi_hess_asv_out ) [private]
```

Coordinates usage of estimate_derivatives() calls based on asv_in.

Splits asv_in total request into map_asv_out, fd_grad_asv_out, fd_hess_asv_out, and quasi_hess_asv_out as governed by the responses specification. If the returned use_est_deriv is true, then these asv outputs are used by estimate_derivatives() for the initial map, finite difference gradient evals, finite difference Hessian evals, and quasi-Hessian updates, respectively. If the returned use_est_deriv is false, then only map_asv_out is used.

References Dakota::abort_handler(), Dakota::contains(), ActiveSet::derivative_vector(), Model::FDstep2(), Dakota::find_index(), Model::forward_grad_step(), Model::gradIdAnalytic, Model::gradIdNumerical, Model::gradientType, Model::hessianType, Model::hessIdAnalytic, Model::hessIdNumerical, Model::hessIdQuasi, Model::ignoreBounds, Model::initialize_x0_bounds(), Model::intervalType, Model::methodSource, Model::model_id(), ActiveSet::request_vector(), Model::shortStep, Model::supportsEstimDerivs, and Model::surrogate_response_mode().

Referenced by Model::evaluate(), and Model::evaluate_nowait().

```cpp
Real initialize_h ( Real x_j, Real lb_j, Real ub_j, Real step_size, String step_type ) const [private]
```

function to determine initial finite difference h (before step length adjustment) based on type of step desired

Auxiliary function to determine initial finite difference h (before step length adjustment) based on type of step desired.

Referenced by Model::estimate_derivatives(), and Model::forward_grad_step().

```cpp
Real FDstep1 ( Real x0_j, Real lb_j, Real ub_j, Real h_mag ) [private]
```

function returning finite-difference step size (affected by bounds)

Auxiliary function to compute forward or first central-difference step size, honoring bounds. The first step is away from zero, when possible. Flips the direction or updates shortStep if can’t take the full requested step h_mag.

References Model::ignoreBounds, and Model::shortStep.

Referenced by Model::estimate_derivatives(), and Model::forward_grad_step().

```cpp
Real FDstep2 ( Real x0_j, Real lb_j, Real ub_j, Real h ) [private]
```

function returning second central-difference step size (affected by bounds)

Auxiliary function to compute the second central-difference step size, honoring bounds.

References Model::ignoreBounds, and Model::shortStep.

Referenced by Model::estimate_derivatives(), and Model::manage_asv().
CHAPTER 13. CLASS DOCUMENTATION

13.83.4 Member Data Documentation

**RealVector fdGradStepSize** [**protected**]
relative finite difference step size for numerical gradients
A scalar value (instead of the vector fd\_gradient\_step\_size spec) is used within the iterator hierarchy since this attribute is only used to publish a step size to vendor numerical gradient algorithms.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::fd\_gradient\_step\_size(), Model::forward\_grad\_step(), RecastModel::initialize\_data\_from\_submodel(), and Model::Model().

**RealVector fdHessByGradStepSize** [**protected**]
relative finite difference step size for numerical Hessians estimated using first-order differences of gradients
For vendor numerical Hessian algorithms, a scalar value is used.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::estimate\_derivatives(), Model::fd\_hessian\_by\_grad\_step\_size(), RecastModel::initialize\_data\_from\_submodel(), and Model::Model().

**RealVector fdHessByFnStepSize** [**protected**]
relative finite difference step size for numerical Hessians estimated using second-order differences of function values
For vendor numerical Hessian algorithms, a scalar value is used.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::estimate\_derivatives(), Model::fd\_hessian\_by\_fn\_step\_size(), RecastModel::initialize\_data\_from\_submodel(), and Model::Model().

**ProblemDescDB& probDescDB** [**protected**]
class member reference to the problem description database

**Iterator** and **Model** cannot use a shallow copy of **ProblemDescDB** due to circular destruction dependency (reference counts can’t get to 0), since **ProblemDescDB** contains \{iterator,model\}List.

Referenced by ActiveSubspaceModel::ActiveSubspaceModel(), NestedModel::derived\_init\_communicators(), HierarchSurrModel::derived\_init\_communicators(), DataFitSurrModel::derived\_init\_communicators(), NestedModel::derived\_init\_serial(), SimulationModel::estimate\_partition\_bounds(), NestedModel::estimate\_partition\_bounds(), HierarchSurrModel::estimate\_partition\_bounds(), DataFitSurrModel::estimate\_partition\_bounds(), Model::problem\_description\_db(), and NestedModel::update\_sub\_iterator().

The documentation for this class was generated from the following files:

- DakotaModel.hpp
- DakotaModel.cpp

13.84 MPIManager Class Reference

Class **MPIManager** to manage Dakota’s MPI world, which may be a subset of MPI\_COMM\_WORLD.

Public Member Functions

- **MPIManager()**
  
  *Default constructor; Dakota will not call MPI\_Init.*

- **MPIManager(int argc, char **argv)**
  
  *Command-line constructor; parses MPI arguments during call to MPI\_Init.*
MPIManager (MPI_Comm dakota_mpi_comm)
Construct on specified MPI_Comm.
~MPIManager ()
destructor: calls finalize if Dakota owns MPI

MPI_Comm dakota_mpi_comm () const
get the MPI_Comm on which Dakota is running

int world_rank () const
get the rank of this process in Dakota’s MPI_Comm

int world_size () const
get the size of the MPI_Comm on which Dakota is running

bool mpirun_flag () const
true when Dakota is running in MPI mode

Static Public Member Functions

static bool detect_parallel_launch (int &argc, char **&argv)
detect parallel launch of Dakota using mpirun/mpiexec/poe/etc. based on command line arguments and environment variables

Private Attributes

MPI_Comm dakotaMPIComm
MPI_Comm on which DAKOTA is running.

int dakotaWorldRank
rank in MPI_Comm in which DAKOTA is running

int dakotaWorldSize
size of MPI_Comm in which DAKOTA is running

bool mpirunFlag
flag for a parallel mpirun/yod launch

bool ownMPIFlag
flag for ownership of MPI_Init/MPI_Finalize

13.84.1 Detailed Description
Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.
The documentation for this class was generated from the following files:

- MPIManager.hpp
- MPIManager.cpp

13.85 MPIPackBuffer Class Reference
Class for packing MPI message buffers.
Public Member Functions

- **MPIPackBuffer (int size_=1024)**
  Constructor, which allows the default buffer size to be set.
- **~MPIPackBuffer ()**
  Destructor.
- **const char * buf ()**
  Returns a pointer to the internal buffer that has been packed.
- **int size ()**
  The number of bytes of packed data.
- **int capacity ()**
  The allocated size of Buffer.
- **void reset ()**
  Resets the buffer index in order to reuse the internal buffer.
- **void pack (const int *data, const int num=1)**
  Pack one or more int’s.
- **void pack (const u_int *data, const int num=1)**
  Pack one or more unsigned int’s.
- **void pack (const long *data, const int num=1)**
  Pack one or more long’s.
- **void pack (const u_long *data, const int num=1)**
  Pack one or more unsigned long’s.
- **void pack (const short *data, const int num=1)**
  Pack one or more short’s.
- **void pack (const u_short *data, const int num=1)**
  Pack one or more unsigned short’s.
- **void pack (const char *data, const int num=1)**
  Pack one or more char’s.
- **void pack (const u_char *data, const int num=1)**
  Pack one or more unsigned char’s.
- **void pack (const double *data, const int num=1)**
  Pack one or more double’s.
- **void pack (const float *data, const int num=1)**
  Pack one or more float’s.
- **void pack (const bool *data, const int num=1)**
  Pack one or more bool’s.
- **void pack (const int &data)**
  Pack a int.
- **void pack (const u_int &data)**
  Pack a unsigned int.
- **void pack (const long &data)**
  Pack a long.
Pack a `unsigned long`.
- void `pack` (const short &data)
- Pack a `short`.
- void `pack` (const u_short &data)
- Pack a `unsigned short`.
- void `pack` (const char &data)
- Pack a `char`.
- void `pack` (const u_char &data)
- Pack a `unsigned char`.
- void `pack` (const double &data)
- Pack a `double`.
- void `pack` (const float &data)
- Pack a `float`.
- void `pack` (const bool &data)
- Pack a `bool`.

Protected Member Functions
- void `resize` (const int newsize)
  Resizes the internal buffer.

Protected Attributes
- char * `Buffer`
  The internal buffer for packing.
- int `Index`
  The index into the current buffer.
- int `Size`
  The total size that has been allocated for the buffer.

13.85.1 Detailed Description
Class for packing MPI message buffers.
A class that provides a facility for packing message buffers using the MPIPack facility. The `MPIPackBuffer` class dynamically resizes the internal buffer to contain enough memory to pack the entire object. When deleted, the `MPIPackBuffer` object deletes this internal buffer. This class is based on the Dakota-Version_3_0 version of utilib::PackBuffer from utilib/src/io/PackBuf.[cpp,h]
The documentation for this class was generated from the following files:
- MPIPackBuffer.hpp
- MPIPackBuffer.cpp

13.86 MPIUnpackBuffer Class Reference
Class for unpacking MPI message buffers.
Public Member Functions

- void setup (char *buf_, int size_, bool flag_=false)
  
  Method that does the setup for the constructors.
- MPIUnpackBuffer ()
  
  Default constructor.
- MPIUnpackBuffer (int size_)
  
  Constructor that specifies the size of the buffer.
- MPIUnpackBuffer (char *buf_, int size_, bool flag_=false)
  
  Constructor that sets the internal buffer to the given array.
- ~MPIUnpackBuffer ()
  
  Destructor.
- void resize (const int newsize)
  
  Resizes the internal buffer.
- const char * buf ()
  
  Returns a pointer to the internal buffer.
- int size ()
  
  Returns the length of the buffer.
- int curr ()
  
  Returns the number of bytes that have been unpacked from the buffer.
- void reset ()
  
  Resets the index of the internal buffer.
- void unpack (int *data, const int num=1)
  
  Unpack one or more int's.
- void unpack (u_int *data, const int num=1)
  
  Unpack one or more unsigned int's.
- void unpack (long *data, const int num=1)
  
  Unpack one or more long's.
- void unpack (u_long *data, const int num=1)
  
  Unpack one or more unsigned long's.
- void unpack (short *data, const int num=1)
  
  Unpack one or more short's.
- void unpack (u_short *data, const int num=1)
  
  Unpack one or more unsigned short's.
- void unpack (char *data, const int num=1)
  
  Unpack one or more char's.
- void unpack (u_char *data, const int num=1)
  
  Unpack one or more unsigned char's.
- void unpack (double *data, const int num=1)
  
  Unpack one or more double's.
- void unpack (float *data, const int num=1)
  
  Unpack one or more float's.
Unpack one or more bool's.
• void unpack (int &data)
  Unpack a int.
• void unpack (u_int &data)
  Unpack a unsigned int.
• void unpack (long &data)
  Unpack a long.
• void unpack (u_long &data)
  Unpack a unsigned long.
• void unpack (short &data)
  Unpack a short.
• void unpack (u_short &data)
  Unpack a unsigned short.
• void unpack (char &data)
  Unpack a char.
• void unpack (u_char &data)
  Unpack a unsigned char.
• void unpack (double &data)
  Unpack a double.
• void unpack (float &data)
  Unpack a float.
• void unpack (bool &data)
  Unpack a bool.

Protected Attributes
• char * Buffer
  The internal buffer for unpacking.
• int Index
  The index into the current buffer.
• int Size
  The total size that has been allocated for the buffer.
• bool ownFlag
  If TRUE, then this class owns the internal buffer.

13.86.1 Detailed Description
Class for unpacking MPI message buffers.
A class that provides a facility for unpacking message buffers using the MPI_Unpack facility. This class is based on the Dakota_Version_3.0 version of utilib::UnPackBuffer from utilib/src/io/PackBuf.[cpp,h]
The documentation for this class was generated from the following files:
• MPIPackBuffer.hpp
• MPIPackBuffer.cpp
13.87 **NCSUTraits Class Reference**

Wrapper class for the NCSU DIRECT optimization library.

Inheritance diagram for NCSUTraits:

```
+ TraitsBase
|   + NCSUTraits

Public Member Functions

- **NCSUTraits ()**
  
  *default constructor*

- **virtual ~NCSUTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

13.87.1 **Detailed Description**

Wrapper class for the NCSU DIRECT optimization library.

The NCSUOptimizer class provides a wrapper for a Fortran 77 implementation of the DIRECT algorithm developed at North Carolina State University. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows: A version of TraitsBase specialized for NCSU optimizers

The documentation for this class was generated from the following file:

- **NCSUOptimizer.hpp**

13.88 **NestedModel Class Reference**

Derived model class which performs a complete sub-iterator execution within every evaluation of the model.

Inheritance diagram for NestedModel:

```
+ Model
|   + NestedModel
```
Public Member Functions

- **NestedModel** (ProblemDescDB &problem_db)
  
  constructor

- **~NestedModel**()
  
  destructor

Protected Member Functions

- void **derived_evaluate** (const ActiveSet &set)
  
  portion of evaluate() specific to NestedModel

- void **derived_evaluate_nowait** (const ActiveSet &set)
  
  portion of evaluate_nowait() specific to NestedModel

- const IntResponseMap & **derived_synchronize**()
  
  portion of synchronize() specific to NestedModel

- Iterator & **subordinate_iterator**()
  
  return subIterator

- Model & **subordinate_model**()
  
  return subModel

- void **derived_subordinate_models** (ModelList &ml, bool recurse_flag)
  
  return subModel

- Interface & **derived_interface**()
  
  return optionalInterface

- const RealVector & **error_estimates**()
  
  retrieve error estimates corresponding to the subIterator’s response results (e.g., statistical MSE for subordinate UQ).

- void **surrogate_response_mode** (short mode)
  
  pass a bypass request on to the subModel for any lower-level surrogates

- void **component_parallel_mode** (short mode)
  
  update component parallel mode for supporting parallelism in optionalInterface and subModel

- size_t **mi_parallel_level_index** () const
  
  return subIteratorSched.miPLIndex

- short **local_eval_synchronization**()
  
  return optionalInterface synchronization setting

- int **local_eval_concurrency**()
  
  return optionalInterface asynchronous evaluation concurrency

- bool **derived_master_overload**() const
  
  flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)

- IntIntPair **estimate_partition_bounds** (int max_eval_concurrency)
  
  estimate the minimum and maximum partition sizes that can be utilized by this Model

- void **derived_init_communicators** (ParLevILiter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  
  set up optionalInterface and subModel for parallel operations

- void **derived_init_serial**()
  
  set up optionalInterface and subModel for serial operations.
• void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration within subModel
• void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the NestedModel (forwarded to optionalInterface and subModel)
• void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)
  Service optionalInterface and subModel job requests received from the master. Completes when a termination
  message is received from stop_servers().
• void stop_servers ()
  Executed by the master to terminate server operations for subModel and optionalInterface when iteration on the
  NestedModel is complete.
• const String & interface_id () const
  return the optionalInterface identifier
• int derived_evaluation_id () const
  Return the current evaluation id for the NestedModel.
• void set_evaluation_reference ()
  set the evaluation counter reference points for the NestedModel (request forwarded to optionalInterface and sub-
  Model)
• void fine_grained_evaluation_counters ()
  request fine-grained evaluation reporting within optionalInterface and subModel
• void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  print the evaluation summary for the NestedModel (request forwarded to optionalInterface and subModel)
• void warm_start_flag (const bool flag)
  set the warm start flag, including actualModel
• void initialize_iterator (int job_index)
• void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
• void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
• void unpack_parameters_initialize (MPIUnpackBuffer &recv_buffer, int job_index)
• void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
• void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
• void update_local_results (int job_index)

Private Member Functions
• void update_sub_iterator ()
  update subIterator with mapping data and set subIterator-based counts
• PRPQueueeliter job_index_to_queue_iterator (int job_index)
  convert job_index to an eval_id through subIteratorIdMap and eval_id to a subIteratorPRPQueue queue iterator
• void initialize_iterator (const Variables &vars, const ActiveSet &set, int eval_id)
  lower level function shared by initialize_iterator(int) and unpack_parameters_initialize()
• void unpack (MPIUnpackBuffer &recv_buffer, int job_index, Variables &vars, ActiveSet &set, int &eval_id)
  lower level function shared by unpack_parameters_buffer() and unpack_parameters_initialize()
• void resolve_real_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
  for a named real mapping, resolve primary index and secondary target
void resolve_integer_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
for a named integer mapping, resolve primary index and secondary target

void resolve_string_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
for a named string mapping, resolve primary index and secondary target

size_t sm_acv_index_map (size_t pacvm_index, short sacvm_target)
offset pacvm_index based on sacvm_target to create mapped_index

size_t sm_adiv_index_map (size_t padivm_index, short sadivm_target)
offset padivm_index based on sadivm_target to create mapped_index

size_t sm_adsv_index_map (size_t padsvm_index, short sadsvm_target)
offset padsvm_index based on sadsvm_target to create mapped_index

size_t sm_adrv_index_map (size_t padrvm_index, short sadrvm_target)
offset padrvm_index based on sadrvm_target to create mapped_index

size_t cv_index_map (size_t cv_index, const Variables &vars)
offset cv_index to create index into aggregated primary/secondary arrays

size_t div_index_map (size_t div_index, const Variables &vars)
offset div_index to create index into aggregated primary/secondary arrays

size_t dsv_index_map (size_t dsv_index, const Variables &vars)
offset dsv_index to create index into aggregated primary/secondary arrays

size_t drv_index_map (size_t drv_index, const Variables &vars)
offset drv_index to create index into aggregated primary/secondary arrays

size_t ccv_index_map (size_t ccv_index, const Variables &vars)
offset active complement ccv_index to create index into all continuous arrays

size_t cdsv_index_map (size_t cdsv_index, const Variables &vars)
offset active complement cdsv_index to create index into all discrete int arrays

size_t cdsv_index_map (size_t cdsv_index, const Variables &vars)
offset active complement cdsv_index to create index into all discrete string arrays

size_t cdsv_index_map (size_t cdsv_index, const Variables &vars)
offset active complement cdsv_index to create index into all discrete real arrays

void real_variable_mapping (const Real &r_var, size_t mapped_index, short svm_target)
insert r_var into appropriate recipient

void integer_variable_mapping (const int &i_var, size_t mapped_index, short svm_target)
insert i_var into appropriate recipient

void string_variable_mapping (const String &s_var, size_t mapped_index, short svm_target)
insert s_var into appropriate recipient

void set_mapping (const ActiveSet &mapped_set, ActiveSet &interface_set, bool &opt_interface_map, ActiveSet &sub_iterator_set, bool &sub_iterator_map)
define the evaluation requirements for the optionalInterface (interface_set) and the subIterator (sub_iterator_set) from the total model evaluation requirements (mapped_set)

void response_mapping (const Response &interface_response, const Response &sub_iterator_response, Response &mapped_response)
combine the response from the optional interface evaluation with the response from the sub-iteration using the primaryCoeffs/secondaryCoeffs mappings to create the total response for the model
CHAPTER 13. CLASS DOCUMENTATION

- void interface_response_overlay (const Response &opt_interface_response, Response &mapped_response)
  assign the response from the optional interface evaluation within the total response for the model
- void iterator_response_overlay (const Response &sub_iterator_response, Response &mapped_response)
  overlay the sub-iteration response within the total response for the model using the primaryCoeffs/secondaryCoeffs mappings
- void iterator_error_estimation (const RealVector &sub_iterator_errors, RealVector &mapped_errors)
  combine error estimates from the sub-iteration to define mappedErrorEstimates
- Response & nested_response (int nested cntr)
  locate existing or allocate new entry in nestedResponseMap
- void check_response_map (const ShortArray &mapped_asv)
  check function counts for the mapped_asv
- void update_inactive_view (short new_view, short &view)
  update inactive variables view for subIterator based on new_view
- void update_inactive_view (unsigned short type, short &view)
  update inactive variables view for subIterator based on type
- void update_sub_model (const Variables &vars, const Constraints &cons)
  update subModel with current variable values/bounds/labels

Private Attributes

- int nestedModelEvalCntr
  number of calls to derived_evaluate()/derived_evaluate_nowait()
- bool firstUpdate
  boolean to trigger one-time updates on first call to update_sub_model()
- IntResponseMap nestedResponseMap
  used to return a map of nested responses (including subIterator and optionalInterface contributions) for aggregation and rekeying at the base class level
- RealVector mappedErrorEstimates
  mapping of subIterator.response_error_estimates() through primary and secondary mappings
- size_t outerMIPLIndex
  the miPLIndex for the outer parallelism context, prior to any subIterator partitioning
- Iterator subIterator
  the sub-iterator that is executed on every evaluation of this model
- String subMethodPointer
  the sub-method pointer from the nested model specification
- Model subModel
  the sub-model used in sub-iterator evaluations
- PRPQueue subIteratorPRPQueue
  job queue for asynchronous execution of subIterator jobs
- IteratorScheduler subIteratorSched
  scheduling object for concurrent iterator parallelism
- int subIteratorJobCntr
  subIterator job counter since last synchronize()
- IntlIntMap subIteratorIdMap
mapping from subIterator evaluation counter to nested model counter (different when subIterator evaluations do not occur on every nested model evaluation due to variable ASV content)

- **size_t numSubIterFns**
  number of sub-iterator response functions prior to mapping

- **size_t numSubIterMappedIneqCon**
  number of top-level inequality constraints mapped from the sub-iteration results

- **size_t numSubIterMappedEqCon**
  number of top-level equality constraints mapped from the sub-iteration results

- **Interface optionalInterface**
  the optional interface contributes nonnested response data to the total model response

- **String optInterfacePointer**
  the optional interface pointer from the nested model specification

- **Response optInterfaceResponse**
  the response object resulting from optional interface evaluations

- **IntIntMap optInterfaceIdMap**
  mapping from optionalInterface evaluation counter to nested model counter (different when optionalInterface evaluations do not occur on every nested model evaluation due to variable ASV content)

- **size_t numOptInterfPrimary**
  number of primary response functions (objective/least squares/generic functions) resulting from optional interface evaluations

- **size_t numOptInterfIneqCon**
  number of inequality constraints resulting from optional interface evaluations

- **size_t numOptInterfEqCon**
  number of equality constraints resulting from the optional interface evaluations

- **SizetArray active1ACVarMapIndices**
  "primary" variable mappings for inserting active continuous currentVariables within all continuous subModel variables. If there are no secondary mappings defined, then the insertions replace the subModel variable values.

- **SizetArray active1ADIVarMapIndices**
  "primary" variable mappings for inserting active discrete int currentVariables within all discrete int subModel variables. No secondary mappings are defined for discrete int variables, so the active variables replace the subModel variable values.

- **SizetArray active1ADSVarMapIndices**
  "primary" variable mappings for inserting active discrete string currentVariables within all discrete string subModel variables. No secondary mappings are defined for discrete string variables, so the active variables replace the subModel variable values.

- **SizetArray active1ADRVarMapIndices**
  "primary" variable mappings for inserting active discrete real currentVariables within all discrete real subModel variables. No secondary mappings are defined for discrete real variables, so the active variables replace the subModel variable values.

- **ShortArray active2ACVarMapTargets**
  "secondary" variable mappings for inserting active continuous currentVariables into sub-parameters (e.g., distribution parameters for uncertain variables or bounds for continuous design/state variables) within all continuous subModel variables.

- **ShortArray active2ADIVarMapTargets**
  "secondary" variable mappings for inserting active discrete int currentVariables into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete int subModel variables.
• **ShortArray** `active2ADSVarMapTargets`
  "secondary" variable mappings for inserting active discrete string `currentVariables` into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete string `subModel` variables.

• **ShortArray** `active2ADRVarMapTargets`
  "secondary" variable mappings for inserting active discrete real `currentVariables` into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete real `subModel` variables.

• **SizetArray** `complement1ACVarMapIndices`
  "primary" variable mappings for inserting the complement of the active continuous `currentVariables` within all continuous `subModel` variables

• **SizetArray** `complement1ADIVarMapIndices`
  "primary" variable mappings for inserting the complement of the active discrete int `currentVariables` within all discrete int `subModel` variables

• **SizetArray** `complement1ADSVarMapIndices`
  "primary" variable mappings for inserting the complement of the active discrete string `currentVariables` within all discrete string `subModel` variables

• **SizetArray** `complement1ADRVarMapIndices`
  "primary" variable mappings for inserting the complement of the active discrete real `currentVariables` within all discrete real `subModel` variables

• **BitArray** `extraCVarsData`
  flags for updating `subModel` continuous bounds and labels, one for each active continuous variable in `currentVariables`

• **BitArray** `extraDIVarsData`
  flags for updating `subModel` discrete int bounds and labels, one for each active discrete int variable in `currentVariables`

• **BitArray** `extraDSVarsData`
  flags for updating `subModel` discrete string labels, one for each active discrete string variable in `currentVariables`

• **BitArray** `extraDRVarsData`
  flags for updating `subModel` discrete real bounds and labels, one for each active discrete real variable in `currentVariables`

• **RealMatrix** `primaryRespCoeffs`
  "primary" response mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level objective functions/least squares/generic response terms.

• **RealMatrix** `secondaryRespCoeffs`
  "secondary" response mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level inequality and equality constraints.

**Friends**

• **class** `IteratorScheduler`
  protect scheduler callback functions from general access
Additional Inherited Members

13.88.1 Detailed Description

Derived model class which performs a complete sub-iterator execution within every evaluation of the model.

The `NestedModel` class nests a sub-iterator execution within every model evaluation. This capability is most commonly used for optimization under uncertainty, in which a nondeterministic iterator is executed on every optimization function evaluation. The `NestedModel` also contains an optional interface, for portions of the model evaluation which are independent from the sub-iterator, and a set of mappings for combining sub-iterator and optional interface data into a top level response for the model.

13.88.2 Member Function Documentation

```cpp
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]
```

portion of `evaluate()` specific to `NestedModel`

- Update subModel’s inactive variables with active variables from currentVariables, compute the optional interface and sub-iterator responses, and map these to the total model response.

  Reimplemented from `Model`.

- References `NestedModel::active2ACVarMapTargets, Response::active_set, ParallelLibrary::bcast, ParallelLibrary::eval_tag, NestedModel::component_parallel_mode, Model::currentResponse, Model::currentVariables, Interface::eval_tag_prefix, Model::evalTagPrefix, Model::hierarchicalTagging, NestedModel::interface_response_overlay, NestedModel::iterator_response_overlay, Iterator::eval_tag_prefix, Model::evalTagPrefix, Model::hierarchicalTagging, NestedModel::interface_response_overlay, NestedModel::iterator_response_overlay, Iterator::eval_tag_prefix, Model::evalTagPrefix, Model::hierarchicalTagging, NestedModel::interface_response_overlay, NestedModel::iterator_response_overlay, Interface::map, IteratorScheduler::iteratorScheduling, Interface::map, IteratorScheduler::messagePass, IteratorScheduler::miPLIndex, Model::modelPCIter, NestedModel::nestedModelEvalCntr, NestedModel::optInterfaceResponse, NestedModel::optionalInterface, Model::parallelLib, IteratorScheduler::peerAssignJobs, Response::reset, Iterator::response_results_active_set, Iterator::run, IteratorScheduler::run_iterator, NestedModel::set_mapping, IteratorScheduler::stop_iterator_servers, NestedModel::subIterator, NestedModel::subIteratorSched, NestedModel::update_sub_model, and Model::userDefinedConstraints.

```cpp
void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]
```

portion of `evaluate_nowait()` specific to `NestedModel`

- Asynchronous execution of subIterator on subModel and, optionally, optionalInterface.

  Reimplemented from `Model`.

- References `Response::active_set, Model::currentResponse, Model::currentVariables, Interface::evaluation_tag_id, Interface::map, Iterator::method_id, NestedModel::nestedModelEvalCntr, NestedModel::optInterfaceIdMap, NestedModel::optInterfaceResponse, NestedModel::optionalInterface, Iterator::response_results_active_set, Iterator::run_iterator, NestedModel::set_mapping, NestedModel::subIterator, NestedModel::subIteratorIdMap, NestedModel::subIteratorJobCntr, and NestedModel::subIteratorPRPQueue.

```cpp
const IntResponseMap & derived_synchronize ( ) [protected], [virtual]
```

portion of `synchronize()` specific to `NestedModel`

- Recovery of asynchronous subIterator executions and, optionally, asynchronous optionalInterface mappings.

  Reimplemented from `Model`.

- References `Interface::cache_unmatched_response, NestedModel::component_parallel_mode, NestedModel::interface_response_overlay, NestedModel::iterator_response_overlay, Model::modelPCIter, NestedModel::nested_response, NestedModel::nestedResponseMap, IteratorScheduler::numIteratorJobs, NestedModel::optInterfaceIdMap, NestedModel::optInterfacePointer, NestedModel::optionalInterface, ParallelLibrary::parallel_configuration-`
short local_eval_synchronization() [inline], [protected], [virtual]

return optionalInterface synchronization setting

Used in setting Model::asynchEvalFlag. subModel synchronization is used for setting asynchEvalFlag within subModel.

Reimplemented from Model.

References Interface::asynch_local_evaluation_concurrency(), Interface::interface_synchronization(), NestedModel::optInterfacePointer, and NestedModel::optionalInterface.

int local_eval_concurrency() [inline], [protected], [virtual]

return optionalInterface asynchronous evaluation concurrency

Used in setting Model::evaluationCapacity. subModel concurrency is used for setting evaluationCapacity within subModel.

Reimplemented from Model.

References Interface::asynch_local_evaluation_concurrency(), NestedModel::optInterfacePointer, and NestedModel::optionalInterface.

bool derived_master_overload() const [inline], [protected], [virtual]

flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)

Derived master overload for subModel is handled separately in subModel.evaluate() within subIterator.run().

Reimplemented from Model.

References Iterator::is_null(), Interface::iterator_eval_dedicated_master(), IteratorScheduler::iteratorScheduling, Interface::multi_proc_eval(), NestedModel::optInterfacePointer, NestedModel::optionalInterface, IteratorScheduler::procsPerIterator, NestedModel::subIterator, and NestedModel::subIteratorSched.

void derived_init_communicators( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag = true ) [protected], [virtual]

set up optionalInterface and subModel for parallel operations

Asynchronous flags need to be initialized for the subModel. In addition, max_eval_concurrency is the outer level iterator concurrency, not the subIterator concurrency that subModel will see, and recomputing the message_lengths on the subModel is probably not a bad idea either. Therefore, recompute everything on subModel using init_communicators().

Reimplemented from Model.

References Response::active_set(), IteratorScheduler::configure(), Model::currentVariables, ProblemDescDB::get_db_method_node(), ProblemDescDB::get_db_model_node(), Interface::init_communicators(), IteratorScheduler::init_iterator(), Iterator::is_null(), IteratorScheduler::iterator_message_lengths(), IteratorScheduler::iteratorServerId, Model::messageLengths, IteratorScheduler::messagePass, Model::modelPCIter, Model::messageLengths, parallel_Library::parallel_configuration_iterator(), Model::parallelLib, IteratorScheduler::partition(), Model::probDescDB, MPIPackBuffer::reset(), Iterator::response_results(), ProblemDescDB::set_db_list_nodes(), ProblemDescDB::set_db_method_node(), ProblemDescDB::set_db_model_nodes(), MPIPackBuffer::size(), NestedModel::subIterator, NestedModel::subModel, NestedModel::subMethodPointer, NestedModel::subModel, IteratorScheduler::update(), and NestedModel::update_sub_iterator().
int derived_evaluation_id ( ) const [inline], [protected], [virtual]

Return the current evaluation id for the NestedModel.

    return the top level nested evaluation count. To get the lower level eval count, the subModel must be explicitly queried. This is consistent with the eval counter definitions in surrogate models.

    Reimplemented from Model.

    References NestedModel::nestedModelEvalCntr.

size_t cv_index_map ( size_t cv_index, const Variables & vars ) [private]

offset cv_index to create index into aggregated primary/secondary arrays maps index within active continuous variables to index within aggregated active continuous/discrete-int/discrete-string/discrete-real variables.

    References SharedVariablesData::aleatory_uncertain_counts(), SharedVariablesData::design_counts(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), Dakota::svd(), and SharedVariablesData::view().

    Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t div_index_map ( size_t div_index, const Variables & vars ) [private]

offset div_index to create index into aggregated primary/secondary arrays maps index within active discrete int variables to index within aggregated active continuous/discrete-int/discrete-string/discrete-real variables.

    References SharedVariablesData::aleatory_uncertain_counts(), Variables::cv(), SharedVariablesData::design_counts(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), SharedVariablesData::state_counts(), Dakota::svd(), and SharedVariablesData::view().

    Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t dsv_index_map ( size_t dsv_index, const Variables & vars ) [private]

offset dsv_index to create index into aggregated primary/secondary arrays maps index within active discrete string variables to index within aggregated active continuous/discrete-int/discrete-string/discrete-string variables.

    References SharedVariablesData::aleatory_uncertain_counts(), Variables::cv(), SharedVariablesData::design_counts(), Variables::div(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), SharedVariablesData::state_counts(), Dakota::svd(), and SharedVariablesData::view().

    Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t drv_index_map ( size_t drv_index, const Variables & vars ) [private]

offset drv_index to create index into aggregated primary/secondary arrays maps index within active discrete real variables to index within aggregated active continuous/discrete-int/discrete-string/discrete-real variables.

    References SharedVariablesData::aleatory_uncertain_counts(), Variables::cv(), SharedVariablesData::design_counts(), Variables::div(), Variables::dsv(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), SharedVariablesData::state_counts(), Dakota::svd(), and SharedVariablesData::view().

    Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().
size_t ccv_index_map ( size_t ccv_index, const Variables & vars ) [private]

offset active complement ccv_index to create index into all continuous arrays
maps index within complement of active continuous variables to index within all continuous variables.
References Dakota::abort_handler(), SharedVariablesData::aleatory_uncertain_counts(), SharedVariablesData::design_counts(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), Dakota::svd(), and
SharedVariablesData::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t cdv_index_map ( size_t cdv_index, const Variables & vars ) [private]

offset active complement cdv_index to create index into all discrete int arrays
maps index within complement of active discrete int variables to index within all discrete int variables.
References Dakota::abort_handler(), SharedVariablesData::aleatory_uncertain_counts(), SharedVariablesData::design_counts(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), Dakota::svd(), and
SharedVariablesData::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t csv_index_map ( size_t csv_index, const Variables & vars ) [private]

offset active complement csv_index to create index into all discrete string arrays
maps index within complement of active discrete string variables to index within all discrete string variables.
References Dakota::abort_handler(), SharedVariablesData::aleatory_uncertain_counts(), SharedVariablesData::design_counts(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), Dakota::svd(), and
SharedVariablesData::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

size_t cdv_index_map ( size_t cdv_index, const Variables & vars ) [private]

offset active complement cdvr_index to create index into all discrete real arrays
maps index within complement of active discrete real variables to index within all discrete real variables.
References Dakota::abort_handler(), SharedVariablesData::aleatory_uncertain_counts(), SharedVariablesData::design_counts(), SharedVariablesData::epistemic_uncertain_counts(), Variables::shared_data(), Dakota::svd(), and
SharedVariablesData::view().
Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

void response_mapping ( const Response & opt_interface_response, const Response & sub_iterator_response, Response & mapped_response ) [inline], [private]

combine the response from the optional interface evaluation with the response from the sub-iteration using the primaryCoeffs/secondaryCoeffs mappings to create the total response for the model

In the OUU case,

optionalInterface fns = {f}, {g} (deterministic primary functions, constraints)
subIterator fns = {S} (UQ response statistics)

Problem formulation for mapped functions:
minimize   {f} + [W]{S}
subject to  {g_l} <= {g} <= {g_u}
{a_l} <= [A]{S} <= {a_u}
{g} -- {g_l}
[A]{S} -- {a_l}
where $[W]$ is the primary mapping matrix user input (primaryRespCoeffs class attribute), $[A]$ is the secondary mapping matrix user input (secondaryRespCoeffs class attribute), $\{\{g_1\},\{a_1\}\}$ are the top level inequality constraint lower bounds, $\{\{g_u\},\{a_u\}\}$ are the top level inequality constraint upper bounds, and $\{\{g_t\},\{a_t\}\}$ are the top level equality constraint targets.

NOTE: optionalInterface/subIterator primary fns (obj/lsq/generic fns) overlap but optionalInterface/subIterator secondary fns (ineq/eq constraints) do not. The $[W]$ matrix can be specified so as to allow

- some purely deterministic primary functions and some combined: $[W]$ filled and $[W].num\_rows() < \{f\}.length() [\text{combined first}]$ or $[W].num\_rows() == \{f\}.length()$ and $[W]$ contains rows of zeros [combined last]
- some combined and some purely stochastic primary functions: $[W]$ filled and $[W].num\_rows() > \{f\}.length()$
- separate deterministic and stochastic primary functions: $[W].num\_rows() > \{f\}.length()$ and $[W]$ contains $\{f\}.length()$ rows of zeros.

If the need arises, could change constraint definition to allow overlap as well: $\{g_1\} <= \{g\} + [A]\{S\} <= \{g_u\}$ with $[A]$ usage the same as for $[W]$ above.

In the UOO case, things are simpler, just compute statistics of each optimization response function: $[W] = [I]$, $\{f\}/[g]/[A]$ are empty.

References Response::active_set_request_vector(), NestedModel::check_response_map(), NestedModel::interface_response_overlay(), and NestedModel::iterator_response_overlay().

### 13.88.3 Member Data Documentation

#### Model subModel

[private]

the sub-model used in sub-iterator evaluations

There are no restrictions on subModel, so arbitrary nestings are possible. This is commonly used to support surrogate-based optimization under uncertainty by having NestedModels contain SurrogateModels and vice versa.

Referenced by NestedModel::component_parallel_mode(), NestedModel::derived_init_communicators(), NestedModel::derived_init_serial(), NestedModel::derived_subordinate_models(), NestedModel::estimate_partition_bounds(), NestedModel::fine_grained_evaluation_counters(), NestedModel::integer_variable_mapping(), NestedModel::NestedModel(), NestedModel::print_evaluation_summary(), NestedModel::real_variable_mapping(), NestedModel::resolve_integer_variable_mapping(), NestedModel::resolve_real_variable_mapping(), NestedModel::resolve_string_variable_mapping(), NestedModel::serve_run(), NestedModel::set_mapping(), NestedModel::sm_acv_index_map(), NestedModel::sm_adv_index_map(), NestedModel::string_variable_mapping(), NestedModel::subordinate_model(), NestedModel::surrogate_response_mode(), NestedModel::update_inactive_view(), NestedModel::update_sub_model(), and NestedModel::warm_start_flag().

The documentation for this class was generated from the following files:

- NestedModel.hpp
- NestedModel.cpp

### 13.89 NIDRProblemDescDB Class Reference

The derived input file database utilizing the new IDR parser.

Inheritance diagram for NIDRProblemDescDB:
CHAPTER 13. CLASS DOCUMENTATION

ProblemDescDB

NIDRProblemDescDB

Public Member Functions

- **NIDRProblemDescDB** (ParallelLibrary &parallel_lib)

  constructor

- **\texttt{\~NIDRProblemDescDB} ()**

  destructor

- void **\texttt{derived\_parse\_inputs}** (const ProgramOptions &prog_opts)

  parses the input file and populates the problem description database using NIDR.

- void **\texttt{derived\_broadcast} ()**

  perform any data processing that must be coordinated with DB buffer broadcasting (performed prior to broadcast-
  ing the DB buffer on rank 0 and after receiving the DB buffer on other processor ranks)

- void **\texttt{derived\_post\_process} ()**

  perform any additional data post-processing

- **\texttt{KWH (iface\_Real)}**

- **\texttt{KWH (iface\_Rlit)}**

- **\texttt{KWH (iface\_false)}**

- **\texttt{KWH (iface\_ilit)}**

- **\texttt{KWH (iface\_pint)}**

- **\texttt{KWH (iface\_lit)}**

- **\texttt{KWH (iface\_start)}**

- **\texttt{KWH (iface\_stop)}**

- **\texttt{KWH (iface\_str)}**

- **\texttt{KWH (iface\_str2D)}**

- **\texttt{KWH (iface\_strL)}**

- **\texttt{KWH (iface\_true)}**

- **\texttt{KWH (iface\_type)}**

- **\texttt{KWH (method\_Ii)}**

- **\texttt{KWH (method\_Real)}**

- **\texttt{KWH (method\_Real01)}**

- **\texttt{KWH (method\_RealDL)}**

- **\texttt{KWH (method\_RealLlit)}**

- **\texttt{KWH (method\_Realp)}**

- **\texttt{KWH (method\_Realz)}**

- **\texttt{KWH (method\_Ri)}**

- **\texttt{KWH (method\_false)}**

- **\texttt{KWH (method\_szarray)}**

- **\texttt{KWH (method\_ilit2)}**

- **\texttt{KWH (method\_ilit2p)}**

- **\texttt{KWH (method\_int)}**
- KWH (method_ivec)
- KWH (method_lit)
- KWH (method_litc)
- KWH (method_litp)
- KWH (method_litr)
- KWH (method_lit)
- KWH (method_order)
- KWH (method_num_resplevs)
- KWH (method_piecewise)
- KWH (method_pint)
- KWH (method_pintz)
- KWH (method_resplevs)
- KWH (method_resplevs01)
- KWH (method_shint)
- KWH (method_sizet)
- KWH (method_slit2)
- KWH (method_start)
- KWH (method_stop)
- KWH (method_str)
- KWH (method_strL)
- KWH (method_true)
- KWH (method_tr_final)
- KWH (method_type)
- KWH (method_usharray)
- KWH (method_ushint)
- KWH (method_utype)
- KWH (method_augment_utype)
- KWH (method_utype_lit)
- KWH (model_Real)
- KWH (model_RealDL)
- KWH (model_ivec)
- KWH (model_false)
- KWH (model_int)
- KWH (model_intsetm1)
- KWH (model_lit)
- KWH (model_order)
- KWH (model_pint)
- KWH (model_shint)
- KWH (model_start)
- KWH (model_stop)
- KWH (model_str)
- KWH (model_strL)
- KWH (model_true)
- KWH (model_type)
• KWH (model_utype)
• KWH (model_augment_utype)
• KWH (resp_RealDL)
• KWH (resp_RealL)
• KWH (resp_false)
• KWH (resp_intset)
• KWH (resp_ivec)
• KWH (resp_lit)
• KWH (resp_siset)
• KWH (resp_start)
• KWH (resp_stop)
• KWH (resp_str)
• KWH (resp_strL)
• KWH (resp_true)
• KWH (resp_utype)
• KWH (resp_augment_utype)
• KWH (env_int)
• KWH (env_start)
• KWH (env_str)
• KWH (env_strL)
• KWH (env_true)
• KWH (env_utype)
• KWH (env_augment_utype)
• KWH (var_RealLb)
• KWH (var_RealUb)
• KWH (var_IntLb)
• KWH (var_categorical)
• KWH (var_ceulbl)
• KWH (var_dauilbl)
• KWH (var_dauslbl)
• KWH (var_deuillbl)
• KWH (var_deuslbl)
• KWH (var_deurlbl)
• KWH (var_pintz)
• KWH (var_start)
• KWH (var_stop)
• KWH (var_str)
• KWH (var_strL)
• KWH (var_true)
• KWH (var_newiarray)
• KWH (var_newsarray)
• KWH (var_newivec)
• KWH (var_newrvec)
• KWH (var_ivec)
• KWH (var_svec)
• KWH (var_rvec)
• KWH (var_type)
13.9. NIDRPROBLEMDESCDB CLASS REFERENCE

Static Public Member Functions

- static void botch (const char *fmt, ...)
  *print and error message and immediately abort*
- static void check_variables (std::list<DataVariables> *)
  *check each node in a list of DataVariables, first mapping DataVariables members back to flat NIDR arrays if needed.*
- static void check_responses (std::list<DataResponses> *)
- static void check_descriptors (const StringArray &labels)
  *Validate user-supplied descriptors.*
- static void make_variable_defaults (std::list<DataVariables> *)
  *Bounds and initial point check and inferred bounds generation.*
- static void make_response_defaults (std::list<DataResponses> *)
- static void squawk (const char *fmt, ...)
  *print an error message and increment nerr, but continue*
- static void warn (const char *fmt, ...)
  *print a warning*

Static Public Attributes

- static NIDRProblemDescDB * pDDBInstance
  *Pointer to the active object instance used within the static kwhandler functions in order to avoid the need for static data. Only initialized when parsing an input file; will be NULL for cases of direct DB population only.*
- static int nerr = 0
  *number of parse error encountered*

Static Private Member Functions

- static void check_variables_node (void *v)
  *check a single variables node; input argument v is Var_Info**
- static int check_driver (const String &an_driver, const StringArray &link_files, const StringArray &copy_files)
  *tokenize and try to validate the presence of an analysis driver, potentially included in the linked or copied template files*

Private Attributes

- std::list< void * > VIL
  *List of Var_Info pointers, one per Variables instance.*

Additional Inherited Members

13.89.1 Detailed Description

The derived input file database utilizing the new IDR parser.

The NIDRProblemDescDB class is derived from ProblemDescDB for use by the NIDR parser in processing DAKOTA input file data. For information on modifying the NIDR input parsing procedures, refer to Dakota/docs/-Dev_Spec_Change.dox. For more on the parsing technology, see "Specifying and Reading Program Input with
NIDR” by David M. Gay (report SAND2008-2261P, which is available in PDF form as http://dakota.sandia.gov/papers/nidr08.pdf). Source for the routines declared herein is NIDRProblemDescDB.cpp, in which most routines are so short that a description seems unnecessary.

13.89.2 Member Function Documentation

```cpp
void derived_parse_inputs ( const ProgramOptions & prog_opts ) [virtual]
```
parses the input file and populates the problem description database using NIDR.

- Parse the input file using the Input Deck Reader (IDR) parsing system. IDR populates the IDRProblemDescDB object with the input file data.
- Reimplemented from ProblemDescDB.

References Dakota::abort_handler(), NIDRProblemDescDB::botch(), ProblemDescDB::dataMethodList, DataMethodRep::dlDetails, DataMethodRep::dlLib, ProgramOptions::input_file(), ProgramOptions::input_string(), NIDRProblemDescDB::nerr, ProblemDescDB::parallel_library(), ProgramOptions::parser_options(), NIDRProblemDescDB::pDDBInstance, and NIDRProblemDescDB::squawk().

```cpp
int check_driver ( const String & an_driver, const StringArray & link_files, const StringArray & copy_files ) [static], [private]
```
tokenize and try to validate the presence of an analysis driver, potentially included in the linked or copied template files

- returns 1 if not found, 2 if found, but not executable, 0 if found (no error) in case we want to return to error on not found...

References WorkdirHelper::find_driver(), NIDRProblemDescDB::squawk(), WorkdirHelper::tokenize_driver(), NIDRProblemDescDB::warn(), and WorkdirHelper::which().

```cpp
void make_variable_defaults ( std::list< DataVariables >∗ dvl ) [static]
```
Bounds and initial point check and inferred bounds generation.

Size arrays for contiguous storage of aggregated uncertain types. For each variable type, call Vgen.* to generate inferred bounds and initial point, repairing initial if needed. size the aggregate arrays for uncertain (design and state are stored separately

References Dakota::DesignAndStateLabelsCheck, NIDRProblemDescDB::squawk(), Dakota::var_mp_drang, Dakota::VLUncertainInt, Dakota::VLUncertainReal, and Dakota::VLUncertainStr.

Referenced by NIDRProblemDescDB::derived_post_process().

The documentation for this class was generated from the following files:

- NIDRProblemDescDB.hpp
- NIDRProblemDescDB.cpp

13.90 NL2Res Struct Reference

Auxiliary information passed to calcr and calcj via ur.

**Public Attributes**

- Real * r

  residual r = r(x)

- Real * J

```
13.91. NL2SOLLEASTSQTRAITS CLASS REFERENCE

\[ \text{Jacobian } J = J(x) \]

- \text{Real } \ast x \]

\text{corresponding parameter vector}

- int \text{nf}

\text{function invocation count for } r(x)

13.90.1 Detailed Description

Auxiliary information passed to calcr and calcj via ur.

The documentation for this struct was generated from the following file:

- NL2SOLLeastSq.cpp

13.91 NL2SOLLeastSqTraits Class Reference

Wrapper class for the NL2SOL nonlinear least squares library.

Inheritance diagram for NL2SOLLeastSqTraits:

```
TraitsBase
  \|-- NL2SOLLeastSqTraits
```

Public Member Functions

- \text{NL2SOLLeastSqTraits ()}

  default constructor

- virtual \text{~NL2SOLLeastSqTraits ()}

  destructor

- virtual bool \text{is\_derived ()}

  \text{A temporary query used in the refactor.}

- bool \text{supports\_continuous\_variables ()}

  \text{Return the flag indicating whether method supports continuous variables.}

13.91.1 Detailed Description

Wrapper class for the NL2SOL nonlinear least squares library.

The NL2SOLLeastSq class provides a wrapper for NL2SOL (TOMS Algorithm 573), in the updated form of Port Library routines dn[fg][b ] from Bell Labs; see \url{http://www.netlib.org/port/readme}. The Fortran from Port has been turned into C by f2c. NL2SOL uses a function pointer approach for which passed functions must be either global functions or static member functions. A version of \text{TraitsBase} specialized for NL2SOL nonlinear least squares library

The documentation for this class was generated from the following file:

- NL2SOLLeastSq.hpp
13.92 NLPQLPTraits Class Reference

Wrapper class for the NLPQLP optimization library, Version 2.0.

Inheritance diagram for NLPQLPTraits:

```
+ TraitsBase
    |    
    ↑    
+ NLPQLPTraits
```

Public Member Functions

- NLPQLPTraits ()
  
  default constructor

- virtual ∼NLPQLPTraits ()
  
  destructor

- virtual bool is_derived ()
  
  A temporary query used in the refactor.

- bool supports_continuous_variables ()
  
  Return the flag indicating whether method supports continuous variables.

- bool supports_linear_equality ()
  
  Return the flag indicating whether method supports linear equalities.

- bool supports_linear_inequality ()
  
  Return the flag indicating whether method supports linear inequalities.

- bool supports_nonlinear_equality ()
  
  Return the flag indicating whether method supports nonlinear equalities.

- bool supports_nonlinear_inequality ()
  
  Return the flag indicating whether method supports nonlinear inequalities.

- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  
  Return the format used for nonlinear inequality constraints.

13.92.1 Detailed Description

Wrapper class for the NLPQLP optimization library, Version 2.0.

AN IMPLEMENTATION OF A SEQUENTIAL QUADRATIC PROGRAMMING
METHOD FOR SOLVING NONLINEAR OPTIMIZATION PROBLEMS BY
DISTRIBUTED COMPUTING AND NON-MONOTONE LINE SEARCH

This subroutine solves the general nonlinear programming problem

\[
\begin{align*}
\text{minimize} & \quad F(X) \\
\text{subject to} & \quad G(J,X) = 0 , \quad J=1,...,ME \\
& \quad G(J,X) \geq 0 , \quad J=ME+1,...,M \\
& \quad XL \leq X \leq XU
\end{align*}
\]
and is an extension of the code NLPQLD. NLPQLP is specifically tuned to run under distributed systems. A new input parameter L is introduced for the number of parallel computers, that is the number of function calls to be executed simultaneously. In case of L=1, NLPQLP is identical to NLPQLD. Otherwise the line search is modified to allow L parallel function calls in advance. Moreover the user has the opportunity to used distributed function calls for evaluating gradients.

The algorithm is a modification of the method of Wilson, Han, and Powell. In each iteration step, a linearly constrained quadratic programming problem is formulated by approximating the Lagrangian function quadratically and by linearizing the constraints. Subsequently, a one-dimensional line search is performed with respect to an augmented Lagrangian merit function to obtain a new iterate. Also the modified line search algorithm guarantees convergence under the same assumptions as before.

For the new version, a non-monotone line search is implemented which allows to increase the merit function in case of instabilities, for example caused by round-off errors, errors in gradient approximations, etc.

The subroutine contains the option to predetermine initial guesses for the multipliers or the Hessian of the Lagrangian function and is called by reverse communication. A version of TraitsBase specialized for NLPQLP optimizers.

The documentation for this class was generated from the following file:

- NLPQLPOptimizer.hpp

13.93 NLSSOLLeastSqTraits Class Reference

Wrapper class for the NLSSOL nonlinear least squares library.

Inheritance diagram for NLSSOLLeastSqTraits:

```plaintext
TraitsBase
   \|-- NLSSOLLeastSqTraits
```

Public Member Functions

- **NLSSOLLeastSqTraits** ()
  
  *default constructor*

- **virtual ~NLSSOLLeastSqTraits** ()
  
  *destructor*

- **virtual bool is_derived** ()
  
  A temporary query used in the refactor.

- **bool supports_continuous_variables** ()
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality** ()
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality** ()
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_equality** ()
  
  *Return the flag indicating whether method supports nonlinear equalities.*
• bool supports_nonlinear_inequality ()

Return the flag indicating whether method supports nonlinear inequalities.

13.93.1 Detailed Description

Wrapper class for the NLSSOL nonlinear least squares library.

The NLSSOLLestSq class provides a wrapper for NLSSOL, a Fortran 77 sequential quadratic programming library from Stanford University marketed by Stanford Business Associates. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any nonstatic attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows: max_function_evaluations is implemented directly in NLSSOLLestSq’s evaluator functions since there is no NLSSOL parameter equivalent, and max.iterations, convergence_tolerance, output verbosity, verify_level, function_precision, and linesearch_tolerance are mapped into NLSSOL’s "Major Iteration Limit", "Optimality Tolerance", "Major Print Level" (verbose: Major Print Level = 20; quiet: Major Print Level = 10), "Verify Level", "Function Precision", and "Linesearch Tolerance" parameters, respectively, using NLSSOL’s npoptn() subroutine (as wrapped by npoptn2() from the npoptn_wrapper.f file). Refer to [Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., 1986] for information on NLSSOL’s optional input parameters and the npoptn() subroutine. A version of TraitsBase specialized for NLSSOL nonlinear least squares library.

The documentation for this class was generated from the following file:

• NLSSOLLestSq.hpp

13.94 NoDBBaseConstructor Struct Reference

Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support.

Public Member Functions

• NoDBBaseConstructor (int=0)

C++ structs can have constructors.

13.94.1 Detailed Description

Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support.

NoDBBaseConstructor is used to overload the constructor used for on-the-fly instantiations in which ProblemDescDB queries cannot be used. Putting this struct here avoids circular dependencies.

The documentation for this struct was generated from the following file:

• dakota_global_defs.hpp

13.95 NomadTraits Class Reference

Wrapper class for NOMAD Optimizer.

Inheritance diagram for NomadTraits:
Public Member Functions

- **NomadTraits ()**
  
  *default constructor*

- **virtual ~NomadTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_discrete_variables ()**
  
  *Return the flag indicating whether method supports discrete variables.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **NONLINEAR_EQUALITY_FORMAT nonlinear_equality_format ()**
  
  *Return the format used for nonlinear equality constraints.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  *Return the format used for nonlinear inequality constraints.*

### 13.95.1 Detailed Description

Wrapper class for NOMAD Optimizer.

NOMAD (is a Nonlinear Optimization by Mesh Adaptive Direct search) is a simulation-based optimization package designed to efficiently explore a design space using Mesh Adaptive Search.

Mesh Adaptive Direct Search uses Meshes, discretizations of the domain space of variables. It generates multiple meshes, and as its name implies, it also adapts the refinement of the meshes in order to find the best solution of a problem.

The objective of each iteration is to find points in a mesh that improves the current solution. If a better solution is not found, the next iteration is done over a finer mesh.

Each iteration is composed of two steps: Search and Poll. The Search step finds any point in the mesh in an attempt to find an improvement; while the Poll step generates trial mesh points surrounding the current best current solution.

The NomadOptimizer is a wrapper for the NOMAD library. It features the following attributes: max_function_evaluations, display_format, display_all_evaluations, function_precision, max_iterations. A version of TraitsBase specialized for Nomad.

The documentation for this class was generated from the following file:

- NomadOptimizer.hpp
13.96 NonD Class Reference

Base class for all nondeterministic iterators (the DAKOTA/UQ branch).

Inheritance diagram for NonD:

```
    NonD
    ├── NonDAnalyzer
    │    ├── NonDIterator
    │    │    ├── NonDCalibration
    │    │    │    ├── NonDExpansion
    │    │    │    │    ├── NonDIntegration
    │    │    │    │    │    ├── NonDInterval
    │    │    │    │    │    │    ├── NonDPOFDarts
    │    │    │    │    │    │    │    ├── NonDReliability
    │    │    │    │    │    │    │    │    ├── NonDRKDDarts
    │    │    │    │    │    │    │    │    │    └── NonDSampling
```

Public Member Functions

- void `initialize_random_variables` (short u_space_type)
  
  initialize natafTransform based on distribution data from iteratedModel

- void `initialize_random_variables` (const Pecos::ProbabilityTransformation &transform, bool deep_copy=false)
  
  alternate form: initialize natafTransform based on incoming data

- void `initialize_random_variable_transformation` ()
  
  instantiate natafTransform

- void `initialize_random_variable_types` ()
  
  initializes runVarTypesX within natafTransform (u-space not needed)

- void `initialize_random_variable_types` (short u_space_type)
  
  initializes runVarTypesX and runVarTypesU within natafTransform

- void `initialize_random_variable_parameters` ()
  
  initializes runVarMeansX, runVarStdDevsX, runVarLowerBndsX, runVarUpperBndsX, and runVarAddtlParamsX within natafTransform
void initialize_random_variable_correlations()
  propagate iteratedModel correlations to natafTransform

void verify_correlation_support (short u_space_type)
  verify that correlation warping is supported by Nataf for given variable types

void transform_correlations()
  perform correlation warping for variable types supported by Nataf

void requested_levels (const RealVectorArray &req_resp_levels, const RealVectorArray &req_prob_levels, const RealVectorArray &req_rel_levels, const RealVectorArray &req_gen_rel_levels, short resp_level_tgt, short resp_level_tgt_reduce, bool cdf_flag, bool pdf_output)
  set requestedRespLevels, requestedProbLevels, requestedRelLevels, requestedGenRelLevels, respLevelTarget, cdfFlag, and pdfOutput (used in combination with alternate ctors)

void distribution_parameter_derivatives (bool dist_param_derivs)
  set distParamDerivs

void print_level_mappings (std::ostream &s) const
  prints the z/p/beta/beta∗ mappings reflected in {requested,computed} {Resp,Prob,Rel,GenRel}Levels for default qoi type and qoi labels

void print_level_mappings (std::ostream &s, String qoi_type, const StringArray &qoi_labels) const
  prints the z/p/beta/beta∗ mappings reflected in {requested,computed} {Resp,Prob,Rel,GenRel}Levels

bool resize()
  reinitializes iterator based on new variable size

bool pdf_output () const
  get pdfOutput

void pdf_output (bool output)
  set pdfOutput

short final_moments_type () const
  get finalMomentsType

void final_moments_type (short type)
  set finalMomentsType

Pecos::ProbabilityTransformation & variable_transformation ()
  return natafTransform

Protected Member Functions

NonD (ProblemDescDB &problem_db, Model &model)
  constructor

NonD (unsigned short method_name, Model &model)
  alternate constructor for sample generation and evaluation “on the fly”

NonD (unsigned short method_name, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation “on the fly”

~NonD ()
  destructor

void derived_set_communicators (ParLevIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance

void initialize_run ()
utility function to perform common operations prior to \texttt{pre\_run()}; typically memory initialization; setting of instance pointers

- \textbf{void pre\_run ()}
  
  \texttt{pre\_run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori}

- \textbf{void finalize\_run ()}
  
  utility function to perform common operations following \texttt{post\_run()}; deallocation and resetting of instance pointers

- \textbf{const Response \& response\_results () const}
  
  return the final statistics from the nondeterministic iteration

- \textbf{void response\_results\_active\_set (const ActiveSet \&set)}
  
  set the active set within finalStatistics

- \textbf{virtual void initialize\_response\_covariance ()}
  
  initializes respCovariance

- \textbf{virtual void initialize\_final\_statistics ()}
  
  initializes finalStatistics for storing NonD final results

- \textbf{virtual void update\_final\_statistics ()}
  
  update finalStatistics::functionValues

- \textbf{void load\_pilot\_sample (const SizetArray \&pilot\_spec, SizetArray \&delta\_N\_l)}
  
  distribute pilot sample specification across model levels

- \textbf{void load\_pilot\_sample (const SizetArray \&pilot\_spec, const Sizet3DArray \&N\_l, SizetArray \&delta\_N\_l)}
  
  distribute pilot sample specification across model forms or levels

- \textbf{void load\_pilot\_sample (const SizetArray \&pilot\_spec, const Sizet3DArray \&N\_l, Sizet2DArray \&delta\_N\_-\_l)}
  
  distribute pilot sample specification across model forms and levels

- \textbf{void size ()}
  
  Size local variables.

- \textbf{int generate\_system\_seed ()}
  
  create a system-generated unique seed (when a seed is unspecified)

- \textbf{void resize\_final\_statistics\_gradients ()}
  
  resizes finalStatistics::functionGradients based on finalStatistics ASV

- \textbf{void update\_aleatory\_final\_statistics ()}
  
  update finalStatistics::functionValues from momentStats and computed\{Prob,Rel,GenRel,Resp\} Levels

- \textbf{void update\_system\_final\_statistics ()}
  
  update system metrics from component metrics within finalStatistics

- \textbf{void update\_system\_final\_statistics\_gradients ()}
  
  update finalStatistics::functionGradients

- \textbf{void initialize\_level\_mappings ()}
  
  size computed\{Resp,Prob,Rel,GenRel\} Levels

- \textbf{void compute\_densities (const RealRealPairArray \&min\_max\_fns, bool prob\_refinement=false, bool all\_levels\_computed=false)}
  
  compute the PDF bins from the CDF/CCDF values and store in computedPDF\{Abscissas,Ordinates\}

- \textbf{void print\_densities (std::ostream \&s) const}
  
  output the PDFs reflected in computedPDF\{Abscissas,Ordinates\} using default qoi\_type and pdf\_labels

- \textbf{void print\_densities (std::ostream \&s, String qoi\_type, const StringArray \&pdf\_labels) const}
output the PDFs reflected in computedPDF\{Abscissas,Ordinates\}

- void \texttt{print\_system\_mappings} (std::ostream &s) const
  
  print system series/parallel mappings for response levels

- void \texttt{print\_multilevel\_evaluation\_summary} (std::ostream &s, const SizetArray &N\_samp)
  
  print evaluation summary for multilevel sampling across 1D profile

- void \texttt{print\_multilevel\_evaluation\_summary} (std::ostream &s, const Sizet2DArray &N\_samp)
  
  print evaluation summary for multilevel sampling across 2D profile

- void \texttt{print\_multilevel\_evaluation\_summary} (std::ostream &s, const Sizet3DArray &N\_samp)
  
  print evaluation summary for multilevel sampling across 3D profile

- void \texttt{transform\_model} (Model &x\_model, Model &u\_model, bool truncated\_bounds=false, Real bound=10.)
  
  recast x\_model from x-space to u-space to create u\_model

- void \texttt{construct\_lhs} (Iterator &u\_space\_sampler, Model &u\_model, unsigned short sample\_type, int num\_samples, int seed, const String &rng, bool vary\_pattern, short sampling\_vars\_mode=ACTIVE)
  
  assign a NonDLHSSampling instance within u\_space\_sampler

- void \texttt{archive\_allocate\_mappings} ()
  
  allocate results array storage for distribution mappings

- void \texttt{archive\_from\_resp} (size_t fn\_index)
  
  archive the mappings from specified response levels for specified fn

- void \texttt{archive\_to\_resp} (size_t fn\_index)
  
  archive the mappings to computed response levels for specified fn

- void \texttt{archive\_allocate\_pdf} ()
  
  allocate results array storage for pdf histograms

- void \texttt{archive\_pdf} (size_t fn\_index)
  
  archive a single pdf histogram for specified function

### Static Protected Member Functions

- static void \texttt{vars\_u\_to\_x\_mapping} (const Variables &u\_vars, Variables &x\_vars)
  
  static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations

- static void \texttt{vars\_x\_to\_u\_mapping} (const Variables &x\_vars, Variables &u\_vars)
  
  static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for NonD Iterators

- static void \texttt{set\_u\_to\_x\_mapping} (const Variables &u\_vars, const ActiveSet &u\_set, ActiveSet &x\_set)
  
  static function for RecastModels used to map u-space ActiveSets from NonD Iterators to x-space ActiveSets for Model evaluations

- static void \texttt{resp\_x\_to\_u\_mapping} (const Variables &x\_vars, const Variables &u\_vars, const Response &x\_-response, Response &u\_response)
  
  static function for RecastModels used to map x-space responses from Model evaluations to u-space responses for return to NonD Iterator.
Protected Attributes

- **NonD** * prevNondInstance
  pointer containing previous value of nondInstance
- **Pecos::ProbabilityTransformation** natafTransform
  Nonlinear variable transformation that encapsulates the required data for performing transformations from X -> Z -> U and back.
- **size_t numContDesVars**
  number of continuous design variables (modeled using uniform distribution for All view modes)
- **size_t numDiscIntDesVars**
  number of discrete integer design variables (modeled using discrete histogram distributions for All view modes)
- **size_t numDiscStringDesVars**
  number of discrete string design variables (modeled using discrete histogram distributions for All view modes)
- **size_t numDiscRealDesVars**
  number of discrete real design variables (modeled using discrete histogram distributions for All view modes)
- **size_t numDesignVars**
  total number of design variables
- **size_t numContStateVars**
  number of continuous state variables (modeled using uniform distribution for All view modes)
- **size_t numDiscIntStateVars**
  number of discrete integer state variables (modeled using discrete histogram distributions for All view modes)
- **size_t numDiscStringStateVars**
  number of discrete string state variables (modeled using discrete histogram distributions for All view modes)
- **size_t numDiscRealStateVars**
  number of discrete real state variables (modeled using discrete histogram distributions for All view modes)
- **size_t numStateVars**
  total number of state variables
- **size_t numNormalVars**
  number of normal uncertain variables (native space)
- **size_t numLognormalVars**
  number of lognormal uncertain variables (native space)
- **size_t numUniformVars**
  number of uniform uncertain variables (native space)
- **size_t numLoguniformVars**
  number of loguniform uncertain variables (native space)
- **size_t numTriangularVars**
  number of triangular uncertain variables (native space)
- **size_t numExponentialVars**
  number of exponential uncertain variables (native space)
- **size_t numBetaVars**
  number of beta uncertain variables (native space)
- **size_t numGammaVars**
  number of gamma uncertain variables (native space)
- **size_t numGumbelVars**
number of gumbel uncertain variables (native space)
- size_t numFrechetVars
  number of frechet uncertain variables (native space)
- size_t numWeibullVars
  number of weibull uncertain variables (native space)
- size_t numHistogramBinVars
  number of histogram bin uncertain variables (native space)
- size_t numPoissonVars
  number of Poisson uncertain variables (native space)
- size_t numBinomialVars
  number of binomial uncertain variables (native space)
- size_t numNegBinomialVars
  number of negative binomial uncertain variables (native space)
- size_t numGeometricVars
  number of geometric uncertain variables (native space)
- size_t numHyperGeomVars
  number of hypergeometric uncertain variables (native space)
- size_t numHistogramPtIntVars
  number of histogram point integer uncertain variables (native space)
- size_t numHistogramPtStringVars
  number of histogram point string uncertain variables (native space)
- size_t numHistogramPtRealVars
  number of histogram point real uncertain variables (native space)
- size_t numContIntervalVars
  number of continuous interval uncertain variables (native space)
- size_t numDiscIntervalVars
  number of discrete interval uncertain variables (native space)
- size_t numDiscSetIntUncVars
  number of discrete integer set uncertain variables (native space)
- size_t numDiscSetStringUncVars
  number of discrete integer set uncertain variables (native space)
- size_t numDiscSetRealUncVars
  number of discrete real set uncertain variables (native space)
- size_t numContAleatUncVars
  total number of continuous aleatory uncertain variables (native space)
- size_t numDiscIntAleatUncVars
  total number of discrete integer aleatory uncertain variables (native space)
- size_t numDiscStringAleatUncVars
  total number of discrete string aleatory uncertain variables (native space)
- size_t numDiscRealAleatUncVars
  total number of discrete real aleatory uncertain variables (native space)
- size_t numAleatoryUncVars
  total number of aleatory uncertain variables (native space)
CHAPTER 13. CLASS DOCUMENTATION

- `size_t numContEpistUncVars`
  total number of continuous epistemic uncertain variables (native space)

- `size_t numDiscIntEpistUncVars`
  total number of discrete integer epistemic uncertain variables (native space)

- `size_t numDiscStringEpistUncVars`
  total number of discrete string epistemic uncertain variables (native space)

- `size_t numDiscRealEpistUncVars`
  total number of discrete real epistemic uncertain variables (native space)

- `size_t numEpistemicUncVars`
  total number of epistemic uncertain variables (native space)

- `size_t numUncertainVars`
  total number of uncertain variables (native space)

- `bool epistemicStats`
  flag for computing interval-type metrics instead of integrated metrics. If any epistemic variables are active in a metric evaluation, then this flag is set.

- `RealMatrix momentStats`
  standardized or central moments of response functions, as determined by finalMomentsType. Calculated in computeMoments() and indexed as (moment,fn).

- `RealVectorArray requestedRespLevels`
  requested response levels for all response functions

- `RealVectorArray computedProbLevels`
  output probability levels for all response functions resulting from requestedRespLevels

- `RealVectorArray computedRelLevels`
  output reliability levels for all response functions resulting from requestedRespLevels

- `RealVectorArray computedGenRelLevels`
  output generalized reliability levels for all response functions resulting from requestedRespLevels

- `short respLevelTarget`
  indicates mapping of z>=p (PROBABILITIES), z>=beta (RELIABILITIES), or z>=beta* (GEN_RELIABILITIES)

- `short respLevelTargetReduce`
  indicates component or system series/parallel failure metrics

- `RealVectorArray requestedProbLevels`
  requested probability levels for all response functions

- `RealVectorArray requestedRelLevels`
  requested reliability levels for all response functions

- `RealVectorArray requestedGenRelLevels`
  requested generalized reliability levels for all response functions

- `RealVectorArray computedRespLevels`
  output response levels for all response functions resulting from requestedProbLevels, requestedRelLevels, or requestedGenRelLevels

- `size_t totalLevelRequests`
  total number of levels specified within requestedRespLevels, requestedProbLevels, and requestedRelLevels

- `bool cdfFlag`
  flag for type of probabilities/reliabilities used in mappings: cumulative/CDF (true) or complementary/CCDF (false)

- `bool pdfOutput`
flag for managing output of response probability density functions (PDFs)

- RealVectorArray `computedPDFAbscissas`
  - sorted response PDF intervals bounds extracted from min/max sample and requested/computedRespLevels (vector lengths = num bins + 1)

- RealVectorArray `computedPDFOrdinates`
  - response PDF densities computed from bin counts divided by (unequal) bin widths (vector lengths = num bins)

- Response `finalStatistics`
  - final statistics from the uncertainty propagation used in strategies: response means, standard deviations, and probabilities of failure

- short `finalMomentsType`
  - type of moments logged within finalStatistics: none, central, standard

- size_t `miPLIndex`
  - index for the active ParallelLevel within ParallelConfiguration::miPLIters

### Static Protected Attributes

- static `NonD * nondInstance`
  - pointer to the active object instance used within static evaluator functions in order to avoid the need for static data

### Private Member Functions

- void `distribute_levels`(RealVectorArray &levels, bool ascending=true)
  - convenience function for distributing a vector of levels among multiple response functions if a short-hand specification is employed.

- void `level_mappings_file`(size_t fn_index, const String &qoi_label) const
  - Write level mappings to a file for a single response.

- void `print_level_map`(std::ostream &s, size_t fn_index, const String &qoi_label) const
  - Print level mapping for a single response function to ostream.

- unsigned short `pecos_to_dakota_variable_type`(unsigned short pecos_var_type)
  - convert from Pecos To Dakota variable enumeration type for continuous aleatory uncertain variables used in variable transformations

- bool `homogeneous`(const SizetArray &N_1) const
  - return true if N_1 has consistent values

### Private Attributes

- bool `distParamDerivs`
  - flags calculation of derivatives with respect to distribution parameters s within `resp_x_to_u_mapping()` using the chain rule df/dx dx/ds. The default is to calculate derivatives with respect to standard random variables u using the chain rule df/dx dx/du.

### 13.96.1 Detailed Description

Base class for all nondeterministic iterators (the DAKOTA/UQ branch).

The base class for nondeterministic iterators consolidates uncertain variable data and probabilistic utilities for inherited classes.
CHAPTER 13. CLASS DOCUMENTATION

13.96.2 Member Function Documentation

void initialize_random_variables ( short u_space_type )

initializes natafTransform based on distribution data from iteratedModel

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

References NonD::initialize_random_variable_correlations(), NonD::initialize_random_variable_parameters(), NonD::initialize_random_variable_transformation(), NonD::initialize_random_variable_types(), and NonD::verify_correlation_support().

Referenced by NonDExpansion::constructExpansion_sampler(), NonDBayesCalibration::construct_mcmc_model(), NonDExpansion::initialize_u_space_model(), NonDAdeptImpSampling::NonDAdeptImpSampling(), NonDGlobalReliability::NonDGlobalReliability(), NonDLocalReliability::NonDLocalReliability(), and NonDLHSSampling::pre_run().

void initialize_random_variables ( const Pecos::ProbabilityTransformation & transform, bool deep_copy = false )

alternate form: initialize natafTransform based on incoming data

This function is commonly used to publish transformation data when the Model variables are in a transformed space (e.g., u-space) and ProbabilityTransformation::ranVarTypes et al. may not be generated directly. This allows for the use of inverse transformations to return the transformed space variables to their original states.

References NonD::initialize_random_variable_transformation(), NonD::natafTransform, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContIntervalVars, and NonD::numContStateVars.

void initialize_random_variable_types ( )

initializes ranVarTypesX within natafTransform (u-space not needed)

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

References Model::aleatory_distribution_parameters(), Model::cv(), Iterator::iteratedModel, NonD::natafTransform, NonD::numBetaVars, NonD::numBinomialVars, NonD::numContDesVars, NonD::numContIntervalVars, NonD::numContStateVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGeometricVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numHyperGeomVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNegBinomialVars, NonD::numNormalVars, NonD::numPoissonVars, NonD::numTriangularVars, NonD::numUniformVars, and NonD::numWeibullVars.

Referenced by NonDExpansion::initialize_random(), NonD::initialize_random_variables(), NonDBayesCalibration::NonDBayesCalibration(), NonDIntegration::NonDIntegration(), NonDReliability::NonDReliability(), and NonD::initialize_random_variable_types().

void initialize_random_variable_types ( short u_space_type )

initializes ranVarTypesX and ranVarTypesU within natafTransform

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Iterator::iteratedModel, NonD::natafTransform, NonD::numBetaVars, NonD::numBinomialVars, NonD::numContDesVars, NonD::numContIntervalVars, NonD::numContStateVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGeometricVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numHyperGeomVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNegBinomialVars, NonD::numNormalVars, NonD::numPoissonVars, NonD::numTriangularVars, NonD::numUniformVars, and NonD::numWeibullVars.

Referenced by NonDExpansion::initialize_random(), NonD::initialize_random_variables(), NonDBayesCalibration::NonDBayesCalibration(), NonDIntegration::NonDIntegration(), NonDReliability::NonDReliability(), and NonD::initialize_random_variable_types().

void initialize_random_variable_types ( short u_space_type )

initializes ranVarTypesX and ranVarTypesU within natafTransform

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Iterator::iteratedModel, NonD::natafTransform, NonD::numBetaVars, NonD::numBinomialVars, NonD::numContDesVars, NonD::numContIntervalVars, NonD::numContStateVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGeometricVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numHyperGeomVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNegBinomialVars, NonD::numNormalVars, NonD::numPoissonVars, NonD::numTriangularVars, NonD::numUniformVars, and NonD::numWeibullVars.
void initialize_random_variable_parameters ( )
initializes ranVarMeansX, ranVarStdDevsX, ranVarLowerBndsX, ranVarUpperBndsX, and ranVarAddtlParamsX within natafTransform
Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.
References Model::aleatory_distribution_parameters(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::epistemic_distribution_parameters(), Iterator::iteratedModel, NonD::natafTransform, NonD::numAleatoryUncVars, NonD::numContDesVars, Analyzer::numContinuousVars, NonD::numContStateVars, and NonD::numEpistemicUncVars.
Referenced by NonDGlobalReliability::core_run(), NonDLocalReliability::core_run(), NonDExpansion::initialize_expansion(), NonDBayesCalibration::initialize_model(), NonD::initialize_random_variables(), and NonDLHS-Sampling::pre_run().

void print_level_mappings ( std::ostream & s, String qoi_type, const StringArray & qoi_labels ) const
prints the z/p/beta/beta∗ mappings reflected in {requested,computed} {Resp,Prob,Rel,GenRel}Levels
Print distribution mappings, including to file per response.
References NonD::level_mappings_file(), Analyzer::numFunctions, Iterator::outputLevel, NonD::print_densities(), NonD::print_level_map(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, and Dakota::write_precision.

void initialize_run ( ) [inline], [protected], [virtual]
utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.
Reimplemented from Analyzer.
References Analyzer::initialize_run(), NonD::nondInstance, and NonD::prevNondInstance.

void pre_run ( ) [inline], [protected], [virtual]
pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
Reimplemented from Analyzer.
Reimplemented in NonDSampling, NonDLHSSampling, NonDRKDDarts, and NonDMultilevelSampling.
References Analyzer::pre_run().
Referenced by NonDRKDDarts::pre_run(), and NonDSampling::pre_run().
void finalize_run( ) [inline], [protected], [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Analyzer::finalize_run(), NonD::nondInstance, and NonD::prevNondInstance.

void initialize_final_statistics( ) [protected], [virtual]

initializes finalStatistics for storing NonD final results

Default definition of virtual function (used by sampling, reliability, and stochastic expansion methods) defines the set of statistical results to include the first two moments and level mappings for each QoI.

Reimplemented in NonDInterval.

References Dakota::abort_handler(), NonD::cdfFlag, Model::cv(), ActiveSet::derivative_vector(), NonD::epistemicStats, NonD::finalMomentsType, NonD::finalStatistics, Response::function_labels(), Model::inactive_continuous_variable_ids(), Iterator::iteratedModel, Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, NonD::respLevelTargetReduce, and NonD::totalLevelRequests.

Referenced by NonDExpansion::NonDExpansion(), NonDIntegration::NonDIntegration(), NonDReliability::NonDReliability(), NonDSampling::NonDSampling(), NonD::requested_levels(), and NonDReliability::resize().

void compute_densities ( const RealRealPairArray & min_max_fns, bool prob_refinement = false, bool all_levels_computed = false ) [protected]

compute the PDF bins from the CDF/CCDF values and store in computedPDF{Abscissas,Ordinates}

This function infers PDFs from the CDF/CCDF level mappings, in order to enable PDF computation after CDF/CCDF probability level refinement (e.g., from importance sampling).

prob_refinement alerts the routine to exclude inverse mappings from the PDF, since refinement only applies to z->p forward mappings and mixing refined and unrefined probability mappings results in an inconsistency (potentially manifesting as negative density values).

all_levels_computed is an option used by reliability methods where computed*Levels are defined across the union of all requested levels.

References NonD::archive_allocate_pdf(), NonD::archive_pdf(), NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedPDFAbscissas, NonD::computedPDFOrdinates, NonD::computedProbLevels, NonD::computedRespLevels, Analyzer::numFunctions, NonD::pdfOutput, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, and NonD::respLevelTarget.

Referenced by NonDExpansion::compute_level_mappings(), NonDExpansion::compute_numerical_statistics(), NonDAdaptImpSampling::core_run(), NonDLocalReliability::core_run(), and NonDGlobalReliability::importance_sampling().

void vars_u_to_x_mapping ( const Variables & u_vars, Variables & x_vars ) [inline], [static], [protected]

static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations

Map the variables from iterator space (u) to simulation space (x).

References Variables::continuous_variables(), Variables::continuous_variables_view(), NonD::natafTransform, and NonD::nondInstance.

Referenced by NonD::transform_model().
void vars_x_to_u_mapping ( const Variables & x_vars, Variables & u_vars ) [inline], [static], [protected]

static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for NonD Iterators

- Map the variables from simulation space (x) to iterator space (u).
- References Variables::continuous_variables(), Variables::continuous_variables_view(), NonD::natafTransform, and NonD::nondInstance.
- Referenced by NonD::transform_model().

void set_u_to_x_mapping ( const Variables & u_vars, const ActiveSet & u_set, ActiveSet & x_set ) [static], [protected]

static function for RecastModels used to map u-space ActiveSets from NonD Iterators to x-space ActiveSets for Model evaluations

- Define the DVV for x-space derivative evaluations by augmenting the iterator requests to account for correlations.
- References Dakota::NPOS, Variables::all_continuous_variable_ids(), Dakota::contains(), Variables::continuous_variable_ids(), ActiveSet::derivative_vector(), Dakota::find_index(), Variables::inactive_continuous_variable_ids(), NonD::natafTransform, and NonD::nondInstance.
- Referenced by NonD::transform_model().

void level_mappings_file ( size_t fn_index, const String & qoi_label ) const [private]

Write level mappings to a file for a single response.
- Write distribution mappings to a file for a single response.
- References NonD::print_level_map(), and Dakota::write_precision.
- Referenced by NonD::print_level_mappings().

void print_level_map ( std::ostream & s, size_t fn_index, const String & qoi_label ) const [private]

Print level mapping for a single response function to ostream.
- Print the distribution mapping for a single response function to the passed output stream. This base class version maps from one requested level type to one computed level type; some derived class implementations (e.g., local and global reliability) output multiple computed level types.
- References NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRelLevels, NonD::computedRespLevels, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and Dakota::write_precision.
- Referenced by NonD::level_mappings_file(), and NonD::print_level_mappings().
- The documentation for this class was generated from the following files:
  - DakotaNonD.hpp
  - DakotaNonD.cpp

13.97 NonDAAdaptImpSampling Class Reference

Class for the Adaptive Importance Sampling methods within DAKOTA.

Inheritance diagram for NonDAAdaptImpSampling:
Public Member Functions

- **NonDAdaptImpSampling** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- **NonDAdaptImpSampling** (Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern, unsigned short is_type, bool cdf Flag, bool x_space_model, bool use_model_bounds, bool track_extreme)

  *alternate constructor for on-the-fly instantiations*

- **~NonDAdaptImpSampling** ()

  *destructor*

- **bool resize** ()

  *reinitializes iterator based on new variable size*

- **void derived_init_communicators** (ParLevLIter pl_iter)

  *derived class contributions to initializing the communicators associated with this Iterator instance*

- **void derived_set_communicators** (ParLevLIter pl_iter)

  *derived class contributions to setting the communicators associated with this Iterator instance*

- **void derived_free_communicators** (ParLevLIter pl_iter)

  *derived class contributions to freeing the communicators associated with this Iterator instance*

- **void core_run** ()

  *performs adaptive importance sampling and computes probability of failure*

- **void print_results** (std::ostream &s, short results_state=FINAL_RESULTS)

  *print the final statistics*

- **void initialize** (const RealVectorArray &full_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)

  *initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations*

- **void initialize** (const RealMatrix &full_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)

  *initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations*

- **void initialize** (const RealVector &full_point, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
initializes data needed for importance sampling: an initial point around which to sample, a failure threshold, an
initial probability to refine, and flags to control transformations

- Real **final_probability** ()
  returns the final probability calculated by the importance sampling

- const RealRealPairArray & **extreme_values** () const
  return extremeValues

### Private Member Functions

- void **select_rep_points** (const RealVectorArray &var_samples_u, const RealVector &fn_samples)
  select representative points from a set of samples

- void **converge_statistics** (bool cov_flag)
  iteratively generate samples and select representative points until probability and (optionally) coefficient of variation converge

- void **generate_samples** (RealVectorArray &var_samples_u)
  generate a set of samples based on multimodal sampling density

- void **evaluate_samples** (const RealVectorArray &var_samples_u, RealVector &fn_samples)
  evaluate the model at the sample points and store the responses

- void **calculate_statistics** (const RealVectorArray &var_samples_u, const RealVector &fn_samples, size_t total_samples, Real &sum_prob, Real &prob, bool compute_cov, Real &sum_var, Real &cov)
  calculate the probability of exceeding the failure threshold and the coefficient of variation (if requested)

- Real **distance** (const RealVector &a, const RealVector &b)
  compute Euclidean distance between points a and b

- Real **recentered_density** (const RealVector &sample_point)
  compute density between a representative point and a sample point, assuming standard normal

### Private Attributes

- Model **uSpaceModel**
  importance sampling is performed in standardized probability space. This u-space model is either passed in (alternate constructor for helper AIS) or constructed using transform_model() (standard constructor for stand-alone AIS)

- unsigned short **importanceSamplingType**
  integration type (is, ais, mmais) provided by input specification

- bool **initLHS**
  flag to identify if initial points are generated from an LHS sample

- bool **useModelBounds**
  flag to control if the sampler should respect the model bounds

- bool **invertProb**
  flag for inversion of probability values using 1.-p

- bool **trackExtremeValues**
  flag for tracking min/max values encountered when evaluating samples

- int **refineSamples**
  size of sample batch within each refinement iteration

- size_t **respFnIndex**
The active response function index in the model to be sampled

- RealVector designPoint
  design subset for which uncertain subset is being sampled

- RealVectorArray initPointsU
  The original set of u-space samples passed in initialize()

- RealVectorArray repPointsU
  The set of representative points in u-space around which to sample

- Real repWeights
  The weight associated with each representative point

- Real probEstimate
  The probability estimate that is iteratively refined by importance sampling

- Real failThresh
  The failure threshold (z-bar) for the problem.

Additional Inherited Members

13.97.1 Detailed Description

Class for the Adaptive Importance Sampling methods within DAKOTA.

The NonDAdaptImpSampling implements the multi-modal adaptive importance sampling used for reliability calculations. (eventually we will want to broaden this). Need to add more detail to this description.

13.97.2 Constructor & Destructor Documentation

NonDAdaptImpSampling ( ProblemDescDB & problem_db, Model & model )

Standard constructor

This is the primary constructor. It accepts a Model reference. It will perform refinement for all response QOI and all probability levels.

References Dakota::abort_handler(), ProblemDescDB::get_iv(), NonD::initialize_random_variables(), Iterator::iteratedModel, NonDSampling::numSamples, Iterator::probDescDB, NonDAdaptImpSampling::refineSamples, NonDSampling::sampleType, NonDSampling::statsFlag, NonD::transform_model(), NonDAdaptImpSampling::useModelBounds, and NonDAdaptImpSampling::uSpaceModel.

NonDAdaptImpSampling ( Model & model, unsigned short sample_type, int refine_samples, int refine_seed, const String & rng, bool vary_pattern, unsigned short is_type, bool cdf_flag, bool x_space_model, bool use_model_bounds, bool track_extreme )

Alternate constructor for on-the-fly instantiations

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB. It will perform refinement for one response QOI and one probability level (passed in initialize()).

References NonD::cdfFlag, NonDSampling::extremeValues, Iterator::maxEvalConcurrency, Analyzer::num-Functions, NonDAdaptImpSampling::refineSamples, NonDAdaptImpSampling::trackExtremeValues, NonD::transform_model(), NonDAdaptImpSampling::useModelBounds, and NonDAdaptImpSampling::uSpaceModel.
13.97.3 Member Function Documentation

```cpp
template <typename RealVector,
          typename RealMatrix,
          typename RealSquareMatrix,
          typename RealVectorSet,
          typename RealVectorArray>

class NonDAdaptiveSampling

private:

public:

    // Constructor
    NonDAdaptiveSampling(const std::string);

    // Destructor
    ~NonDAdaptiveSampling();

    // Initialize
    void initialize(const RealVectorArray &acv_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)

    // Initialize using a vector array of starting points
    void initialize(const RealVectorArray &acv_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)

    // Initialize using a matrix of starting points
    void initialize(const RealMatrix &acv_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)

    // Initialize using only one starting point
    void initialize(const RealVector &acv_point, bool x_space_data, size_t resp_index,Real initial_prob, Real failure_threshold)

private:

    // Member variables
    RealVectorArray acv_points;
    bool x_space_data;
    size_t resp_index;
    Real initial_prob;
    Real failure_threshold;

    // Member functions
    void initialize(const RealVectorArray &acv_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
```

Initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations.

Initializes data using a vector array of starting points.

Initializes data using a matrix of starting points.

Initializes data using only one starting point.

The documentation for this class was generated from the following files:

- NonDAdaptImpSampling.hpp
- NonDAdaptImpSampling.cpp

13.98 NonDAdaptiveSampling Class Reference

Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

Inheritance diagram for NonDAdaptiveSampling:
Public Member Functions

- **NonDAdaptiveSampling** *(ProblemDescDB &problem_db, Model &model)*
  - A standard constructor
- **~NonDAdaptiveSampling** ()
  - An alternate constructor for sample generation and evaluation “on the fly” has not been implemented
- **bool resize ()**
  - Reinitializes iterator based on new variable size

Protected Member Functions

- **void derived_init_communicators** (ParLevLIter pl_iter)
  - Derived class contributions to initializing the communicators associated with this Iterator instance
- **void derived_set_communicators** (ParLevLIter pl_iter)
  - Derived class contributions to setting the communicators associated with this Iterator instance
- **void derived_free_communicators** (ParLevLIter pl_iter)
  - Derived class contributions to freeing the communicators associated with this Iterator instance
- **void core_run ()**
  - Core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- **Real final_probability ()**
- **void print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  - Print the final iterator results

Private Member Functions

- **void calc_score_alm ()**
  - Function to compute the ALM scores for the candidate points. ALM score is the variance computed by the surrogate at the point.
- **void calc_score_delta_x ()**
  - Function to compute the Distance scores for the candidate points. Distance score is the shortest distance between the candidate and an existing training point.
- **void calc_score_delta_y ()**
Function to compute the Gradient scores for the candidate points. Gradient score is the function value difference between a candidate’s surrogate response and its nearest evaluated true response from the training set.

- **void calc_score_topo_bottleneck ()**
  Function to compute the Bottleneck scores for the candidate points. Bottleneck score is computed by determining the bottleneck distance between the persistence diagrams of two approximate Morse-Smale complices. The complices used include one built from only the training data, and another built from the training data and the single candidate.

- **void calc_score_topo_avg_persistence (int respFnCount)**
  Function to compute the Average Change in Persistence scores for the candidate points. Avg Persistence score is computed as the average change in persistence each point undergoes between two approximate Morse-Smale complices. The complices used include one built from only the training data, and another built from the training data and the single candidate.

- **void calc_score_topo_highest_persistence (int respFnCount)**
  Function to compute the Highest Persistence scores for the candidate points. Highest Persistence score is calculated as a ranking of a set of candidates by constructing an approximate Morse-Smale complex over the entire set of candidates, using their surrogate responses, and the training data, using their true responses, and ranking points based on the most topological significance as measured by their persistence values. In the case where there are no topologically significant points, the point will be chosen randomly. TODO: It may be wiser to fall back to a scheme that ranks points based on proximity to extrema, or the most significant extrema?

- **void calc_score_topo_alm_hybrid (int respFnCount)**
  Function to compute the Hybrid scores for the candidate points. Hybrid score is computed the same as Avg Persistence score except that instead of computing one score, three scores are computing not only a mean surface, but a mean +/- std. dev. surfaces and then averaging the three separate scores. The hope is that you strike a balance between selecting points in topologically important areas and areas of high uncertainty.

- **Real calc_score_alm (int respFnCount, RealVector &test_point)**
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- **Real calc_score_delta_x (int respFnCount, RealVector &test_point)**
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- **Real calc_score_delta_y (int respFnCount, RealVector &test_point)**
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- **Real calc_score_topo_bottleneck (int respFnCount, RealVector &test_point)**
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- **Real calc_score_topo_avg_persistence (int respFnCount, RealVector &test_point)**
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- **Real calc_score_topo_alm_hybrid (int respFnCount, RealVector &test_point)**
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- **Real compute_rmspe ()**
  Using the validationSet, compute the RMSE over the surface.

- **void compare_complices (int dim, std::ostream &output)**
  Using the validationSet, compute the approximate Morse-Smale complices of the true model over the validationSet as well as the surrogate model over the validationSet, and output some topological comparisons.

- **void parse_options ()**
  Parse misc options specified in a user input deck.

- **RealVectorArray drawNewX (int this_k, int respFnCount=0)**
  Function to pick the next X value to be evaluated by the Iterated model.

- **void output_round_data (int round, int respFnCount=0)**
  Temporary function for dumping validation data to output files to be visualized in TopoAS.
void update_amsa (int respFnCount=0)

Update the approximate Morse-Smale complex based on the training points and selected candidates. Uses surrogate function responses.

void construct_fsu_sampler (Iterator &u_space_sampler, Model &u_model, int num_samples, int seed, unsigned short sample_type)

Copy of construct_lhs only it allows for the construction of FSU sample designs. This can break the fsu_cvt, so it is not used at the moment, and these designs only affect the initial sample build not the candidate sets constructed at each round.

void output_for_optimization (int dim)

This function will write an input deck for a multi-start global optimization run of DAKOTA by extracting all of the local minima off the approximate Morse-Smale complex created from the validation set of the surrogate model.

Real median (const RealVector &sorted_data)

calculate the median of the sorted values passed in

void pick_new_candidates ()

Pick new candidates from Emulator.

void score_new_candidates ()

Score new candidates based on the chosen metrics.

Private Attributes

- Iterator gpBuild
  LHS iterator for building the initial GP.
- Iterator gpEval
  LHS iterator for sampling on the GP.
- Iterator gpFinalEval
  LHS iterator for sampling on the final GP.
- Model gpModel
  GP model of response, one approximation per response function.
- int numRounds
  the number of rounds of additions of size batchSize to add to the original set of LHS samples
- int numPtsTotal
  the total number of points
- int numEmulEval
  the number of points evaluated by the GP each iteration
- int numFinalEmulEval
  number of points evaluated on the final GP
- int scoringMethod
  the type of scoring metric to use for sampling
- Real finalProb
  the final calculated probability (p)
- RealVectorArray gpCvars
  Vector to hold the current values of the current sample inputs on the GP.
- RealVectorArray gpMeans
  Vector to hold the current values of the current mean estimates for the sample values on the GP.
• RealVectorArray gpVar
  Vector to hold the current values of the current variance estimates for the sample values on the GP.
• RealVector emulEvalScores
  Vector to hold the scored values for the current GP samples.
• RealVector predictionErrors
  Vector to hold the RMSE after each round of adaptively fitting the model.
• RealVectorArray validationSet
  Validation point set used to determine predictionErrors above.
• RealVector yTrue
  True function responses at the values corresponding to validationSet.
• RealVector yModel
  Surrogate function responses at the values corresponding to validationSet.
• int validationSetSize
  Number of points used in the validationSet.
• int batchSize
  Number of points to add each round, default = 1.
• String batchStrategy
  String describing the type of batch addition to use. Allowable values are naive, distance, topology.
• String outputDir
  Temporary string for dumping validation files used in TopoAS visualization.
• String scoringMetric
  String describing the method for scoring candidate points. Options are: alm, distance, gradient, highest_persistence, avg_persistence, bottleneck, alm_topo_hybrid. Note: alm and alm_topo_hybrid will fail when used with surrogates other than global_kriging as it is based on the variance of the surrogate. At the time of implementation, global_kriging is the only surrogate capable of yielding this information.
• unsigned short sampleDesign
  Enum describing the initial sample design. Options are: RANDOM_SAMPLING, FSU_CVT, FSU_HALTON, FSU_HAMMERSLEY.
• String approx_type
  String describing type of surrogate is used to fit the data. Options are: global_kriging, global_mars, global_neural_network, global_polynomial, global_moving_least_squares, global_radial_basis.
• MS_Complex * AMSC
  The approximate Morse-Smale complex data structure.
• int numKneighbors
  The number of approximate nearest neighbors to use in computing the AMSC.
• bool outputValidationData
  Temporary variable for toggling writing of data files to be used by TopoAS.

Additional Inherited Members

13.98.1 Detailed Description

Class for testing various adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

NonDAdaptiveSampling implements an adaptive sampling method based on the work presented in Adaptive Sampling with Topological Scores by Dan Maljovec, Bei Wang, Ana Kupresanin, Gardar Johannesson, Valerio
Pascucci, and Peer-Timo Bremer presented in IJUQ (insert issue). The method computes scores based on the topology of the known data and the topology of the surrogate model. A number of alternate adaption strategies are offered as well.

13.98.2 Constructor & Destructor Documentation

NonDAdaptiveSampling (ProblemDescDB & problem_db, Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), Response::active_set(), NonDAdaptiveSampling::AMSC, NonDAdaptiveSampling::approx_type, Iterator::assign_rep(), Model::assign_rep(), NonDAdaptiveSampling::batchSize, NonDAdaptiveSampling::batchStrategy, NonDAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), Model::current_response(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_iv(), ProblemDescDB::get_s(), ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), NonDAdaptiveSampling::gpBuild, NonDAdaptiveSampling::gpEval, NonDAdaptiveSampling::gpFinalEval, NonDAdaptiveSampling::gpModel, Model::gradient_type(), Model::hessian_type(), Iterator::iteratedModel, Iterator::maxIterations, NonDAdaptiveSampling::numEmulEval, NonDAdaptiveSampling::numFinalEmulEval, NonDAdaptiveSampling::numKneighbors, NonDAdaptiveSampling::numRounds, NonDAdaptiveSampling::numSamples, NonDAdaptiveSampling::outputDir, Iterator::outputLevel, NonDAdaptiveSampling::outputValidationData, NonDAdaptiveSampling::parse_options(), Iterator::probDescDB, NonDSampling::randomSeed, ActiveSet::request_values(), NonDSampling::rngName, NonDAdaptiveSampling::sampleDesign, NonDSampling::sampleType, NonDAdaptive Sampling::scoringMetric, NonDAdaptiveSampling::scoringMetric, NonDAdaptiveSampling::update_amsc().

~NonDAdaptiveSampling ( )

alternate constructor for sample generation and evaluation "on the fly" has not been implemented

destructor

13.98.3 Member Function Documentation

void core_run ( ) [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Iterator::all_responses(), Model::append_approximation(), Model::approximation_data(), NonDAdaptiveSampling::batchSize, Model::build_approximation(), NonDAdaptiveSampling::compare_complinces(), NonDAdaptiveSampling::compute_rmspe(), NonD::computedProbLevels, Model::continuous_variables(), Model::current_response(), Model::current_variables(), NonDAdaptiveSampling::drawNewX(), Model::evaluate(), Model::evaluation_id(), NonDAdaptiveSampling::gpCvars, NonDAdaptiveSampling::gpFinalEval, NonDAdaptiveSampling::gpMeans, NonDAdaptiveSampling::gpModel, NonDAdaptiveSampling::gpVar, NonD::initialize_level_mappings(), Iterator::iteratedModel, Iterator::methodPCIter, NonD::miPLIndex, NonDAdaptiveSampling::numEmulEval, NonDAdaptiveSampling::numFinalEmulEval, NonDAdaptiveSampling::numKneighbors, NonDAdaptiveSampling::numRounds, NonDAdaptiveSampling::numSamples, NonDAdaptiveSampling::output_round_data(), NonDAdaptiveSampling::pick_new_candidates(), NonDAdaptiveSampling::predictionErrors, NonD::requestedRespLevels, Iterator::run(), NonDAdaptiveSampling::score_new_candidates(), NonDAdaptiveSampling::scoringMetric, and NonDAdaptiveSampling::update_amsc().
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
Reimplemented from Analyzer.
References NonD::print_level_mappings(), and NonDSampling::statsFlag.
The documentation for this class was generated from the following files:
• NonDAdaptiveSampling.hpp
• NonDAdaptiveSampling.cpp

13.99 NonDBayesCalibration Class Reference
Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.
Inheritance diagram for NonDBayesCalibration:

Public Member Functions
• NonDBayesCalibration (ProblemDescDB &problem_db, Model &model)
  standard constructor
• ~NonDBayesCalibration ()
  destructor
• template<typename VectorType>
  Real prior_density (const VectorType &vec)
  compute the prior PDF for a particular MCMC sample
• template<typename VectorType>
  Real log_prior_density (const VectorType &vec)
  compute the log prior PDF for a particular MCMC sample
• template<typename Engine >
  void prior_sample (Engine &rng, RealVector &prior_samples)
  draw a multivariate sample from the prior distribution

• template<typename VectorType >
  void prior_mean (VectorType &mean_vec) const
  return the mean of the prior distribution

• template<typename MatrixType >
  void prior_variance (MatrixType &var_mat) const
  return the covariance of the prior distribution

Static Public Member Functions

• static void get_positive_definite_covariance_from_hessian (const RealSymMatrix &hessian, const RealMatrix &prior_chol_fact, RealSymMatrix &covariance, short output_lev)
  Compute the proposal covariance C based on low-rank approximation to the prior-preconditioned misfit Hessian.

• static Real knn_kl_div (RealMatrix &distX_samples, RealMatrix &distY_samples, size_t dim)
• static Real knn_mutual_info (RealMatrix &Xmatrix, int dimX, int dimY, int alg)

Protected Member Functions

• void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
• void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
• void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
• void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
• void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results
• const Model & algorithm_space_model () const
• virtual void calibrate ()=0
  Perform Bayesian calibration (all derived classes must implement)
• void construct_mcmc_model ()
  construct mcmcModel (no emulation, GP, PCE, or SC) that wraps inbound Model
• void init_hyper_parameters ()
  initialize the hyper-parameter priors
• void construct_map_optimizer ()
  construct the negative log posterior RecastModel (wraps residualModel) and corresponding MAP optimizer
• void initialize_model ()
  initialize emulator model and probability space transformations
• void calibrate_to_hifi ()
  calibrate the model to a high-fidelity data source, using mutual information-guided design of experiments (adaptive experimental design)
• void build_model_discrepancy ()
calculate model discrepancy with respect to experimental data

- void calculate_kde ()
  calculate a Kernel Density Estimate (KDE) for the posterior samples

- void extract_selected_posterior_samples (const std::vector<int> &points_to_keep, const RealMatrix &samples_for_posterior_eval, const RealVector &posterior_density, RealMatrix &posterior_data) const

- void export_posterior_samples_to_file (const std::string filename, const RealMatrix &posterior_data) const

- template<typename VectorType1 , typename VectorType2 >
  void augment_gradient_with_log_prior (VectorType1 &log_grad, const VectorType2 &vec)
  compute the (approximate) gradient of the negative log posterior by augmenting the (approximate) gradient of the negative log likelihood with the gradient of the negative log prior

- template<typename MatrixType , typename VectorType >
  void augment_hessian_with_log_prior (MatrixType &log_hess, const VectorType &vec)
  compute the (approximate) Hessian of the negative log posterior by augmenting the (approximate) Hessian of the negative log likelihood with the Hessian of the negative log prior

- Real log_likelihood (const RealVector &residuals, const RealVector &hyperparams)
  calculate log-likelihood from the passed residuals (assuming they are already sized and scaled by covariance / hyperparams...)

- void prior_cholesky_factorization ()
  compute priorCovCholFactor based on prior distributions for random variables and any hyperparameters

- void get_positive_definite_covariance_from_hessian (const RealSymMatrix &hessian, RealSymMatrix &covariance)

- void export_discrepancy (RealMatrix &pred_config_mat)
  print tabular files containing model+discrepancy responses and variances

- void compute_statistics ()
  Compute final stats for MCMC chains.

- void export_chain ()
  export the acceptance chain in user space

- void export_chain (RealMatrix &filtered_chain, RealMatrix &filtered_fn_vals)
  Print filtered posterior and function values (later: credibility and prediction intervals)

- void filter_chain (RealMatrix &acceptance_chain, RealMatrix &filtered_chain)
  Perform chain filtering with burn-in and sub-sampling.

- void filter_fnvals (RealMatrix &accepted_fn_vals, RealMatrix &filtered_fn_vals)

- void compute_intervals ()

- void compute_prediction_vals (RealMatrix &filtered_fn_vals, RealMatrix &PredVals, int num_filter, size_t num_concatenated)

- void compute_col_means (RealMatrix &matrix, RealVector &avg_vals)

- void compute_col_stdevs (RealMatrix &matrix, RealVector &avg_vals, RealVector &std_devs)

- void print_intervals_file (std::ostream &stream, RealMatrix &functionvalsT, RealMatrix &predvalsT, int length, size_t aug_length)

- void print_intervals_screen (std::ostream &stream, RealMatrix &functionvalsT, RealMatrix &predvalsT, int length)

- void kl_post_prior (RealMatrix &acceptanceChain)
  Compute information metrics.

- void prior_sample_matrix (RealMatrix &prior_dist_samples)

- void mutual_info_buildX ()

- void print_kl (std::ostream &stream)
Static Protected Member Functions

- static void neg_log_post_resp_mapping (const Variables &model_vars, const Variables &nlpost_vars, const Response &model_resp, Response &nlpost_resp)
  static function passed by pointer to negLogPostModel recast model
- static void ann_dist (const ANNpointArray matrix1, const ANNpointArray matrix2, RealVector &distances, int NX, int NY, int dim2, IntVector &k, double eps)
- static void ann_dist (const ANNpointArray matrix1, const ANNpointArray matrix2, RealVector &distances, Int2DArray &indices, int NX, int NY, int dim2, IntVector &k, double eps)

Protected Attributes

- String scalarDataFilename
- short emulatorType
  the emulator type: NO_EMULATOR, GP_EMULATOR, PCE_EMULATOR, SC_EMULATOR, ML_PCE_EMULATOR, MF_PCE_EMULATOR, or MF_SC_EMULATOR
- Model mcmcModel
  Model instance employed in the likelihood function; provides response function values from Gaussian processes, stochastic expansions (PCE/SC), or direct access to simulations (no surrogate option)
- bool mcmcModelHasSurrogate
  whether the MCMC Model is a surrogate, or a thin transformation around a surrogate, so can be cheaply re-evaluated in chain recovery
- Model residualModel
  DataTransformModel wrapping the mcmcModel.
- Iterator mapOptimizer
  SQP or NIP optimizer for pre-solving for the MAP point prior to MCMC. This is restricted to emulator cases for now, but as for derivative preconditioning, could be activated for no-emulator cases with a specification option (not active by default).
- Model negLogPostModel
  RecastModel for solving for MAP: reduces residualModel to scalar negative log posterior.
- unsigned short mapOptAlgOverride
  user setting of the MAP optimization algorithm type
- Iterator stochExpIterator
  NonDPolynomialChaos or NonDStochCollocation instance for defining a PCE/SC-based mcmcModel.
- int chainSamples
  number of samples in the chain (e.g. number of MCMC samples); for iterative update cycles, number of samples per update cycle
- int chainCycles
  number of update cycles for MCMC chain (implemented by restarting of short chains)
- int randomSeed
  random seed for MCMC process
- short mcmcDerivOrder
  order of derivatives used in MCMC process (bitwise like ASV)
- bool adaptExpDesign
  whether to perform iterative design of experiments with high-fidelity model
- size_t numCandidates
- String `importCandPtsFile`  
  whether to import candidate design points for adaptive Bayesian experimental design
- unsigned short `importCandFormat`  
  tabular format for the candidate design points import file
- int `maxHifiEvals`  
  maximum number of high-fidelity model runs to be used for adaptive Bayesian experimental design
- bool `mutualInfoKSG2`  
  whether the KSG2 algorithm is to be employed in the calculation of the mutual information
- bool `calModelDiscrepancy`  
  flag whether to calculate model discrepancy
- String `discrepancyType`  
  set discrepancy type
- String `exportCorrModelFile`  
  filename for corrected model (model+discrepancy) calculations output
- String `exportDiscrepFile`  
  filename for discrepancy calculations output
- String `exportCorrVarFile`  
  filename for corrected model variance calculations
- unsigned short `exportCorrModelFormat`  
  format options for corrected model output
- unsigned short `exportDiscrepFormat`  
  format options for discrepancy output
- unsigned short `exportCorrVarFormat`  
  format options for corrected model variance output
- short `approxCorrectionOrder`  
  specify polynomial or trend order for discrepancy correction
- size_t `numPredConfigs`  
  number of prediction configurations at which to calculate model discrepancy
- RealVector `configLowerBnds`  
  lower bounds on configuration domain
- RealVector `configUpperBnds`  
  upper bounds on configuration domain
- ResponseArray `discrepancyResponses`  
  array containing predicted of model+discrepancy
- ResponseArray `correctedResponses`  
  array containing predicted of model+discrepancy
- RealMatrix `correctedVariances`  
  matrix containing variances of model+discrepancy
- RealVector `predictionConfigList`  
  list of prediction configurations at which to calculate model discrepancy
- String `importPredConfigs`  
  whether to import prediction configurations at which to calculate model discrepancy
• unsigned short **importPredConfigFormat**
  
  *tabular format for prediction configurations import file*

• **Model hifiModel**
  
  *a high-fidelity model data source (given by pointer in input)*

• int **initHifiSamples**
  
  *initial high-fidelity model samples*

• **Iterator hifiSampler**
  
  *LHS iterator to generate hi-fi model data.*

• **RealMatrix priorCovCholFactor**
  
  *the Cholesky factor of the prior covariance*

• unsigned short **obsErrorMultiplierMode**
  
  *mode for number of observation error multipliers to calibrate (default none)*

• int **numHyperparams**
  
  *calculated number of hyperparameters augmenting the calibration parameter set, e.g., due to calibrate observation error multipliers*

• **RealVector invGammaAlphas**
  
  *alphas for inverse gamma distribution on hyper-params*

• **RealVector invGammaBetas**
  
  *alphas for inverse gamma distribution on hyper-params*

• std::vector
  
  < Pecos::RandomVariable > **invGammaDists**
  
  *distributions for hyper-params*

• **bool standardizedSpace**
  
  *flag indicating use of a variable transformation to standardized probability space for the model or emulator*

• **bool posteriorStatsKL**
  
  *flag indicating the calculation of KL divergence between prior and posterior*

• **bool posteriorStatsMutual**
  
  *flag indicating the calculation of mutual information between prior and posterior*

• **bool adaptPosteriorRefine**
  
  *flag indicating usage of adaptive posterior refinement; currently makes sense for unstructured grids in GP and PCE least squares/CS*

• **String proposalCovarType**
  
  *approach for defining proposal covariance*

• **RealVector proposalCovarData**
  
  *data from user input of proposal covariance*

• **String proposalCovarFilename**
  
  *filename for user-specified proposal covariance*

• **String proposalCovarInputType**
  
  *approach for defining proposal covariance*

• **RealMatrix acceptanceChain**
  
  *Post-processing-related controls.*

• **RealMatrix acceptedFnVals**
  
  *cached function values corresponding to acceptanceChain for final statistics reporting*
• std::map<Real, RealVector> bestSamples
  container for managing best MCMC samples (points and associated log posterior) collected across multiple (restarted) chains
• int burnInSamples
  number of MCMC samples to discard from acceptance chain
• int subSamplingPeriod
  period or skip in post-processing the acceptance chain
• RealMatrix chainStats
• RealMatrix fnStats
• RealMatrix predVals
  Compute credibility and prediction intervals of final chain.
• RealMatrix filteredFnVals
  cached filtered function values for printing
• String exportMCMCFilename
  output filename for the MCMC chain
• short exportMCMCFormat
  output formatting options for MCMC export
• short filteredMCMCFormat
• Real kl_est

Static Protected Attributes
• static NonDBayesCalibration * nonDBayesInstance
  Pointer to current class instance for use in static callback functions.

13.99.1 Detailed Description
Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.
This class will eventually provide a general-purpose framework for Bayesian inference. In the short term, it only collects shared code between QUESO and GPMSA implementations.

13.99.2 Constructor & Destructor Documentation
NonDBayesCalibration ( ProblemDescDB & problem_db, Model & model )
standard constructor
This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.
References Dakota::abort_handler(), NonDBayesCalibration::adaptExpDesign, Iterator::assign_rep(), Model::assign_rep(), NonDCalibration::calibrationData, NonDBayesCalibration::construct_map_optimizer(), NonDBayesCalibration::construct_mcmc_model(), Model::continuous_lower_bound(), Model::continuous_upper_bound(), NonDBayesCalibration::emulatorType, NonDCalibration::expData, NonD::generate_system_seed(), ProblemDescDB::get_bool(), NonDBayesCalibration::hifiModel, NonDBayesCalibration::hifiSampler, NonDBayesCalibration::init_hyper_parameters(), NonDBayesCalibration::initHifiSamples, NonD::initialize_random_variable_correlations(),
NonD::initialize_random_variable_transformation(), NonD::initialize_random_variable_types(), Iterator::iterated-Model, Iterator::maxEvalConcurrency, NonDBayesCalibration::mcmcDerivOrder, NonDBayesCalibration::mcmc-Model, Model::model_type(), ExperimentData::num_experiments(), Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, Iterator::probDescDB, NonDBayesCalibration::randomSeed, NonDBayesCalibration::residualModel, NonDBayesCalibration::standardizedSpace, Model::truth_model(), Analyzer::vary_pattern(), and NonD::verify_correlation_support().

13.99.3 Member Function Documentation

void prior_mean ( VectorType & mean_vec ) const

return the mean of the prior distribution
Assume the target mean_vec is sized by client
References NonDBayesCalibration::invGammaDists, NonD::natafTransform, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, and NonDBayesCalibration::standardizedSpace.

void prior_variance ( MatrixType & var_mat ) const

return the covariance of the prior distribution
Assumes the target var_mat is sized by client
References NonDBayesCalibration::invGammaDists, NonD::natafTransform, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, and NonDBayesCalibration::standardizedSpace.

void core_run ( ) [protected], [virtual]
core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References NonDBayesCalibration::adaptExpDesign, NonDBayesCalibration::build_model_discrepancy(), NonDBayesCalibration::calibrate(), NonDBayesCalibration::calibrate_to_hifi(), NonDBayesCalibration::calModelDiscrepancy, and NonDBayesCalibration::nonDBayesInstance.

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
Reimplemented from Analyzer.
Reimplemented in NonDGPMSABayesCalibration, NonDWASABIBayesCalibration, and NonDQUESTOBayesCalibration.
References Model::continuous_variable_labels(), Dakota::copy_data(), Model::current_response(), NonDBayesCalibration::filteredFnVals, Response::function_labels(), NonDBayesCalibration::mcmcModel, Iterator::output-Level, NonDBayesCalibration::posteriorStatsKL, NonDSampling::print_moments(), NonD::requestedProbLevels, and NonDBayesCalibration::residualModel.
References by NonDQUESTOBayesCalibration::print_results(), and NonDGPMSABayesCalibration::print_results().
const Model & algorithm_space_model() const  [inline], [protected], [virtual]

default definition that gets redefined in selected derived Minimizers
  Reimplemented from Analyzer.
  References NonDBayesCalibration::mcmcModel.

void construct_map_optimizer() [protected]

construct the negative log posterior RecastModel (wraps residualModel) and corresponding MAP optimizer
  Construct optimizer for MAP pre-solve Emulator: on by default; can be overridden with "pre_solve none" No emulator: off by default; can be activated with "pre_solve {sqp,nip}" relies on mapOptimizer ctor to enforce min derivative support
  References Iterator::assign_rep(), Model::assign_rep(), Iterator::convergenceTol, NonDBayesCalibration::emulator- Type, Iterator::gnewton_set_recast(), NonDBayesCalibration::mapOptAlgOverride, NonDBayesCalibration::map- Optimizer, NonDBayesCalibration::mcmcDerivOrder, NonDBayesCalibration::neg_log_post_resp_mapping(), NonDBayesCalibration::negLogPostModel, Model::num_primary_fns(), and NonDBayesCalibration::residualModel.
  Referenced by NonDBayesCalibration::calibrate_to_hifi(), and NonDBayesCalibration::NonDBayesCalibration().

Real log_likelihood ( const RealVector & residuals, const RealVector & all_params ) [protected]
calculate log-likelihood from the passed residuals (assuming they are already sized and scaled by covariance / hyperparams...)
  Calculate the log-likelihood, accounting for contributions from covariance and hyperparameters, as well as constant term:
  \[
  \log(L) = \frac{-1}{2} \cdot nr \cdot \log(2 \cdot \pi) - \frac{1}{2} \cdot \log(\det(Cov)) - \frac{1}{2} \cdot r'(Cov^{-1}) \cdot r
  \]
  The passed residuals must already be size-adjusted, differenced with any data, if present, and scaled by covariance^{-1/2}.
  References NonDCalibration::expData, Dakota::HALF_LOG_2PI, ExperimentData::half_log_cov_determinant(), Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, and NonDBayesCalibration::obsErrorMultiplierMode.
  Referenced by NonDQUESOBayesCalibration::dakotaLogLikelihood(), NonDBayesCalibration::neg_log_post_resp_mapping(), and NonDDREAMBayesCalibration::sample_likelihood().

void neg_log_post_resp_mapping ( const Variables & residual_vars, const Variables & nlpost_vars, const Response & residual Resp, Response & nlpost Resp ) [static], [protected]

static function passed by pointer to negLogPostModel recast model
  Response mapping callback used within RecastModel for MAP pre-solve. Computes
  \[
  -\log(\text{post}) = -\log(\text{like}) - \log(\text{prior}); \text{ where } -\log(\text{like}) = \frac{1}{2} \cdot nr \cdot \log(2 \cdot \pi) + \frac{1}{2} \cdot \log(\det(Cov)) + \frac{1}{2} \cdot r'(Cov^{-1}) \cdot r
  \]
  (misfit defined as 1/2 r^T (mult^{-2}Gamma_d^{-1}) r) The passed residual Resp has been differenced, interpolated, and covariance-scaled
  References Response::active_set_request_vector(), NonDBayesCalibration::augment_gradient_with_log_prior(), NonDBayesCalibration::augment_hessian_with_log_prior(), ExperimentData::build_gradient_of_sum_square_residuals(), ExperimentData::build_hessian_of_sum_square_residuals(), Variables::continuous_variables(), NonDCalibration::expData, Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), Response::function_values(), ExperimentData::half_log_cov_det_gradient(), ExperimentData::half_log_cov_det_hessian(), NonDBayesCalibration::log_likelihood(), NonDBayesCalibration::log_prior_density(), NonDBayesCalibration::non- DBayesInstance, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, and Iterator::outputLevel.
  Referenced by NonDBayesCalibration::construct_map_optimizer().
void export_chain ( RealMatrix & filtered_chain, RealMatrix & filtered_fn_vals ) [protected]

Print filtered posterior and function values (later: credibility and prediction intervals)
  Print tabular file with filtered chain, function values, and pred values
  References Variables::continuous_variables(), Variables::copy(), Model::current_response(), Model::current_variables(), NonDBayesCalibration::exportMCMCFilename, NonDBayesCalibration::exportMCMCFormat, Response::function_labels(), Model::interface_id(), NonDBayesCalibration::mcmcModel, Analyzer::numFunctions, NonDBayesCalibration::residualModel, Dakota::write_precision, and Variables::write_tabular().

13.99.4 Member Data Documentation

RealMatrix acceptanceChain [protected]

Post-processing-related controls.
  accumulation of acceptance chain across restarts (stored in user-space) TO DO: retire once restarts are retired; optimize to convert to user-space only in final results
  Referenced by NonDQUESOBayesCalibration::aggregate_acceptance_chain(), NonDDREAMBayesCalibration::archive_acceptance_chain(), NonDBayesCalibration::build_model_discrepancy(), NonDGPMSABayesCalibration::cache_acceptance_chain(), NonDREAMBayesCalibration::cache_chain(), NonDBayesCalibration::compute_statistics(), and NonDQUE-SOBayesCalibration::run_chain_with_restarting().

The documentation for this class was generated from the following files:
  • NonDBayesCalibration.hpp
  • NonDBayesCalibration.cpp

13.100 NonDCalibration Class Reference

Inheritance diagram for NonDCalibration:
Public Member Functions

- **NonDCalibration** (ProblemDescDB &problem db, Model &model)
  
  *standard constructor*

- **~NonDCalibration** ()
  
  *destructor*

- **bool resize** ()
  
  *reinitializes iterator based on new variable size*

Protected Attributes

- **bool calibrationData**
  
  *flag indicating whether there is calibration data present*

- **ExperimentData expData**
  
  *Container for experimental data to which to calibrate model.*

Additional Inherited Members

13.100.1 Detailed Description

This class ...

13.100.2 Constructor & Destructor Documentation

**NonDCalibration** ( ProblemDescDB & problem db, Model & model )

*standard constructor*

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References NonDCalibration::calibrationData, NonDCalibration::expData, ExperimentData::load_data(), and Iterator::outputLevel.

The documentation for this class was generated from the following files:

- NonDCalibration.hpp
- NonDCalibration.cpp

13.101 NonDCubature Class Reference

Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

Inheritance diagram for NonDCubature:
Public Member Functions

- `NonDCubature (Model &model, const Pecos::ShortArray &u_types, unsigned short cub_int_order)`
- `unsigned short integrand_order () const`
  
  ```
  return cubIntOrder
  ```

Protected Member Functions

- `NonDCubature (ProblemDescDB &problem_db, Model &model)`
  
  ```
  constructor
  ```

- `~NonDCubature ()`
  
  ```
  destructor
  ```

- `void initialize_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)`
  
  ```
  initialize integration grid by drawing from polynomial basis settings
  ```

- `void get_parameter_sets (Model &model)`
  
  ```
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
  ```

- `void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)`

- `void increment_grid ()`
  
  ```
  increment SSG level/TPQ order
  ```

- `void increment_grid_preference (const RealVector &dim_pref)`

- `int num_samples () const`

Private Member Functions

- `void check_integration (const Pecos::ShortArray &u_types, const Pecos::AleatoryDistParams &adp)`
  
  ```
  verify self-consistency of integration specification
  ```

- `void increment_reference ()`
  
  ```
  increment each cubIntOrderRef entry by 1
  ```
Private Attributes

- Pecos::CubatureDriver * cubDriver
  
  convenience pointer to the numIntDriver representation

- unsigned short cubIntOrderRef
  
  reference point for Pecos::CubatureDriver::cubIntOrder: the original user specification for the number of Gauss points per dimension, plus any refinements posted by increment_grid()

- unsigned short cubIntRule
  
  the isotropic cubature integration rule

Additional Inherited Members

13.101.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of moments. It employs Stroud cubature rules and extensions by D. Xiu.

13.101.2 Constructor & Destructor Documentation

NonDCubature ( Model & model, const Pecos::ShortArray & u_types, unsigned short cub_int_order )

This alternate constructor is used for on-the-fly generation and evaluation of numerical cubature points.

References Model::aleatory_distribution_parameters(), NonDCubature::check_integration(), NonDCubature::cubDriver, NonDCubature::cubIntOrderRef, Iterator::iteratedModel, and NonDIntegration::numIntDriver.

NonDCubature ( ProblemDescDB & problem_db, Model & model ) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate nond_cubature method specification.

References Model::aleatory_distribution_parameters(), NonDCubature::check_integration(), NonDIntegration::check_variables(), NonDCubature::cubDriver, NonDCubature::cubIntOrderRef, NonDCubature::cubIntRule, Iterator::iteratedModel, Iterator::maxEvalConcurrency, NonD::natafTransform, and NonDIntegration::numIntDriver.

13.101.3 Member Function Documentation

void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the cubature routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDCubature::cubDriver, and NonDCubature::cubIntOrderRef.
void increment_grid_preference ( const RealVector & dim_prefer ) [inline], [protected], [virtual]

Should not be used, but pure virtual must be defined.
Reimplemented from NonDIntegration.
References NonDCubature::increment_grid().

int num_samples ( ) const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.
Reimplemented from Analyzer.
References NonDCubature::cubDriver.

void increment_reference ( ) [inline], [private]

increment each cubIntOrderRef entry by 1
hungIntOrderRef is a reference point for CubatureDriver::cubIntOrder, e.g., a lower bound
References NonDCubature::cubIntOrderRef.
Referenced by NonDCubature::increment_grid().
The documentation for this class was generated from the following files:

- NonDCubature.hpp
- NonDCubature.cpp

13.102 NonDDREAMBayesCalibration Class Reference

Bayesian inference using the DREAM approach.

Inheritance diagram for NonDDREAMBayesCalibration:

```
NonDDREAMBayesCalibration
    NonDBayesCalibration
        NonDCalibration
            NonD
                Analyzer
                    Iterator
```

Public Member Functions

- NonDDREAMBayesCalibration (ProblemDescDB &problem_db, Model &model)
  standard constructor
• ~\texttt{NonDDREAMBayesCalibration} ()
  
  destructor

\textbf{Static Public Member Functions}

• static void \texttt{problem\_size} (int \&chain\_num, int \&cr\_num, int \&gen\_num, int \&pair\_num, int \&par\_num)
  
  \textit{initializer for problem size characteristics in DREAM}

• static void \texttt{problem\_value} (std::string \*chain\_filename, std::string \*gr\_filename, double \&gr\_threshold, int \&jumpstep, double limits[], int par\_num, int \&printstep, std::string \*restart\_read\_filename, std::string \*restart\_write\_filename)
  
  \textit{Filename and data initializer for DREAM.}

• static double \texttt{prior\_density} (int par\_num, double zp[])
  
  \textit{Compute the prior density at specified point zp.}

• static double \* \texttt{prior\_sample} (int par\_num)
  
  \textit{Sample the prior and return an array of parameter values.}

• static double \texttt{sample\_likelihood} (int par\_num, double zp[])
  
  \textit{Likelihood function for call-back from DREAM to DAKOTA for evaluation.}

\textbf{Protected Member Functions}

• void \texttt{calibrate} ()

• void \texttt{archive\_acceptance\_chain} ()
  
  \textit{save the final x-space acceptance chain and corresponding function values}

\textbf{Static Protected Member Functions}

• static void \texttt{cache\_chain} (const double *const z)
  
  \textit{Callback to archive the chain from DREAM, potentially leaving it in u-space.}

\textbf{Protected Attributes}

• RealVector \texttt{paramMins}
  
  \textit{lower bounds on calibrated parameters}

• RealVector \texttt{paramMaxs}
  
  \textit{upper bounds on calibrated parameters}

• int \texttt{numChains}
  
  \textit{number of concurrent chains}

• int \texttt{numGenerations}
  
  \textit{number of generations}

• int \texttt{numCR}
  
  \textit{number of CR-factors}

• int \texttt{crossoverChainPairs}
  
  \textit{number of crossover chain pairs}

• Real \texttt{grThreshold}
  
  \textit{threshold for the Gelmin-Rubin statistic}
CHAPTER 13. CLASS DOCUMENTATION

- int jumpStep
  
  how often to perform a long jump in generations

- boost::mt19937 numGenerator
  
  random number engine for sampling the prior

Static Private Attributes

- static NonDDREAMBayesCalibration * nonDDREAMInstance
  
  Pointer to current class instance for use in static callback functions.

Additional Inherited Members

13.102.1 Detailed Description

Bayesian inference using the DREAM approach.

This class performed Bayesian calibration using the DREAM (Markov Chain Monte Carlo acceleration by Differential Evolution) implementation of John Burkhardt (FSU), adapted from that of Guannan Zhang (ORNL)

13.102.2 Constructor & Destructor Documentation

NonDDREAMBayesCalibration ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References NonDBayesCalibration::chainSamples, NonDDREAMBayesCalibration::crossoverChainPairs, NonDDREAMBayesCalibration::grThreshold, NonDDREAMBayesCalibration::jumpStep, NonDDREAMBayesCalibration::numChains, NonDDREAMBayesCalibration::numCR, and NonDDREAMBayesCalibration::numGenerations.

13.102.3 Member Function Documentation

void problem_size ( int & chain_num, int & cr_num, int & gen_num, int & pair_num, int & par_num )

[static]

initializer for problem size characteristics in DREAM

See documentation in DREAM examples

References NonDDREAMBayesCalibration::crossoverChainPairs, NonDDREAMBayesCalibration::nonDDREAMBayesCalibration::numChains, Analyzer::numContinuousVars, NonDDREAMBayesCalibration::numCR, NonDDREAMBayesCalibration::numGenerations, and NonDBayesCalibration::numHyperparams.

void problem_value ( std::string * chain_filename, std::string * gr_filename, double & gr_threshold, int & jumpstep, double limits[], int par_num, int & printstep, std::string * restart_read_filename, std::string * restart_write_filename )

[static]

Filename and data initializer for DREAM.

See documentation in DREAM examples

References NonDDREAMBayesCalibration::grThreshold, NonDDREAMBayesCalibration::jumpStep, NonDDREAMBayesCalibration::nonDDREAMBayesInstance, NonDDREAMBayesCalibration::numChains, NonDDREAMBayesCalibration::paramMaxs, and NonDDREAMBayesCalibration::paramMins.
double prior_density ( int par_num, double zp[] ) [static]

Compute the prior density at specified point zp.
   See documentation in DREAM examples
   References NonDBayesCalibration::nonDBayesInstance, and NonDBayesCalibration::prior_density().

double * prior_sample ( int par_num ) [static]

Sample the prior and return an array of parameter values.
   See documentation in DREAM examples
   References NonDBayesCalibration::nonDBayesInstance, NonDDREAMBayesCalibration::nonDDREAMInstance, and NonDDREAMBayesCalibration::rnumGenerator.

double sample_likelihood ( int par_num, double zp[] ) [static]

Likelihood function for call-back from DREAM to DAKOTA for evaluation.
   Static callback function to evaluate the likelihood
   References Model::continuous_variables(), Model::current_response(), Model::evaluate(), Response::function_values(), NonDBayesCalibration::log_likelihood(), NonDDREAMBayesCalibration::nonDDREAMInstance, Iterator::outputLevel, and NonDBayesCalibration::residualModel.

void calibrate ( ) [protected], [virtual]

Perform the uncertainty quantification DREAM will callback to cache_chain to store the chain
   Implements NonDBayesCalibration.
   References Dakota::abort_handler(), NonDDREAMBayesCalibration::archive_acceptance_chain(), NonDDREAMBayesCalibration::cache_chain(), NonDCalibration::calibrationData, NonDBayesCalibration::chainSamples, NonDBayesCalibration::compute_statistics(), Model::continuous_variables(), NonDBayesCalibration::initialize_model(), NonDBayesCalibration::mcmcModel, NonD::natafTransform, NonDDREAMBayesCalibration::nonDDREAMInstance, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, NonDDREAMBayesCalibration::paramMaxs, NonDDREAMBayesCalibration::paramMins, NonDBayesCalibration::randomSeed, NonDDREAMBayesCalibration::rnumGenerator, and NonDBayesCalibration::standardizedSpace.

void cache_chain ( const double *const z ) [static], [protected]

Callback to archive the chain from DREAM, potentially leaving it in u-space.
   Archive the chain from DREAM. This default implementation is aggregating from the parallel chains in a round-robin fashion.
   References NonDBayesCalibration::acceptanceChain, NonDDREAMBayesCalibration::nonDDREAMInstance, Analyzer::num_samples(), NonDDREAMBayesCalibration::numChains, Analyzer::numContinuousVars, NonDDREAMBayesCalibration::numGenerations, and NonDBayesCalibration::numHyperparams.
   Referenced by NonDDREAMBayesCalibration::calibrate().
   The documentation for this class was generated from the following files:
   * NonDDREAMBayesCalibration.hpp
   * NonDDREAMBayesCalibration.cpp
13.103 NonDExpansion Class Reference

Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC)

Inheritance diagram for NonDExpansion:

```
  NonDExpansion
    |     
    v     
  NonD
    |     
    v     
  NonDExpansion
    |     
    v     
  NonDPolynomialChaos  NonDStochCollocation
    |     
    v     
  NonDMultilevelPolynomialChaos  NonDMultilevelStochCollocation
```

Public Member Functions

- **NonDExpansion** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- **NonDExpansion** (unsigned short method_name, Model &model, short exp_coeffs_approach, bool piecewise_basis, bool use_derivs)

  *alternate constructor*

- **~NonDExpansion** ()

  *destructor*

- **bool resize ()**

  *reinitializes iterator based on new variable size*

- **void derived_init_communicators** (ParLevLIter pl_iter)

  *derived class contributions to initializing the communicators associated with this Iterator instance*

- **void derived_set_communicators** (ParLevLIter pl_iter)

  *derived class contributions to setting the communicators associated with this Iterator instance*

- **void derived_free_communicators** (ParLevLIter pl_iter)

  *derived class contributions to freeing the communicators associated with this Iterator instance*

- **void core_run ()**

  *perform a forward uncertainty propagation using PCE/SC methods*

- **void print_results** (std::ostream &s, short results_state=FINAL_RESULTS)

  *print the final statistics*

- **const Model & algorithm_space_model () const**

- **virtual void select_refinement_points** (const RealVectorArray &candidate_samples, unsigned short batch_size, RealMatrix &best_samples)

  *evaluate allSamples for inclusion in the (PCE regression) approximation and retain the best set (well spaced) of size batch_size*
virtual void append_expansion (const RealMatrix &samples, const IntResponseMap &resp_map, size_t index=NPOS)
append new data to uSpaceModel and update expansion order (PCE only)

Protected Member Functions

virtual void resolve_inputs (short &u_space_type, short &data_order)
perform error checks and mode overrides

virtual void initialize_u_space_model ()
initialize uSpaceModel polynomial approximations with PCE/SC data

virtual void initialize_expansion ()
initialize random variable definitions and final stats arrays

virtual void compute_expansion (size_t index=NPOS)
form the expansion by calling uSpaceModel.build_approximation()

virtual void increment_order_and_grid ()
uniformly increment the expansion order and structured/unstructured grid (PCE only)

virtual void increment_specification_sequence ()
increment the input specification sequence (PCE only)

virtual void update_expansion (size_t index=NPOS)
update an expansion; avoids overhead in compute_expansion()

virtual void archive_coefficients ()
archive expansion coefficients, as supported by derived instance

virtual Real compute_covariance_metric ()
compute 2-norm of change in response covariance

virtual Real compute_final_statistics_metric ()
compute 2-norm of change in final statistics

void initialize_response_covariance ()
set covarianceControl defaults and shape respCovariance

void update_final_statistics ()
update function values within finalStatistics

void update_final_statistics_gradients ()
update function gradients within finalStatistics

void initialize_random (short u_space_type)
common constructor code for initialization of natafTransform

void check_dimension_preference (const RealVector &dim_pref) const
check length and content of dimension preference vector

void assign_hierarchical_response_mode ()
define the surrogate response mode for a hierarchical model in multilevel/multifidelity expansions

void construct_cubature (Iterator &u_space_sampler, Model &g_u_model, unsigned short cub_int_order)
assign a NonDCubature instance within u_space_sampler

void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, unsigned short quad_order, const RealVector &dim_pref)
assign a NonDQuadrature instance within u_space_sampler based on a quad_order specification
CHAPTER 13. CLASS DOCUMENTATION

- **void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, unsigned short quad_order, const RealVector &dim_pref, int filtered_samples)**
  assign a NonDQuadrature instance within u_space_sampler that generates a filtered tensor product sample set
- **void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, unsigned short quad_order, const RealVector &dim_pref, int random_samples, int seed)**
  assign a NonDQuadrature instance within u_space_sampler that samples randomly from a tensor product multi-index
- **void construct_sparse_grid (Iterator &u_space_sampler, Model &g_u_model, unsigned short ssg_level, const RealVector &dim_pref)**
  assign a NonDSparseGrid instance within u_space_sampler
- **void construct_expansion_sampler (const String &import_approx_file, unsigned short import_approx_format=TABLEULAR_ANOTATED, bool import_approx_active_only=false)**
  construct the expansionSampler for evaluating samples on uSpaceModel
- **void multifidelity_expansion ()**
  construct a multifidelity expansion, across model forms or discretization levels
- **void refine_expansion (size_t index=NPOS)**
  refine the reference expansion found by compute_expansion() using uniform/adaptive p-h-refinement strategies
- **void compute_statistics (short results_state=FINAL_RESULTS)**
  calculate analytic and numerical statistics from the expansion
- **void annotated_results (short results_state=FINAL_RESULTS)**
  manage computing of results and debugging outputs, for either the final set of results (full statistics) or an intermediate set of results (reduced set of core results used for levels/fidelities, etc.)
- **void archive_moments ()**
  archive the central moments (numerical and expansion) to ResultsDB

Protected Attributes

- **Model uSpaceModel**
  Model representing the approximate response function in u-space, after u-space recasting and polynomial data fit recursions.
- **short expansionCoeffsApproach**
  method for collocation point generation and subsequent calculation of the expansion coefficients
- **short multilevDiscrepEmulation**
  emulation approach for multilevel discrepancy: distinct or recursive
- **short expansionBasisType**
  type of expansion basis: DEFAULT_BASIS or Pecos::[NODAL,HIERARCHICAL],INTERPOLANT for SC or Pecos::[TENSOR_PRODUCT,TOTAL_ORDER,ADAPTED]_BASIS for PCE regression
- **size_t numUncertainQuant**
  number of invocations of core_run()
- **int numSamplesOnModel**
  number of truth samples performed on g_u_model to form the expansion
- **int numSamplesOnExpansion**
  number of approximation samples performed on the polynomial expansion in order to estimate probabilities
- **bool nestedRules**

flag for indicating state of nested and non_nested overrides of default rule nesting, which depends on the type of integration driver

- **bool piecewiseBasis**
  flag for piecewise specification, indicating usage of local basis polynomials within the stochastic expansion

- **bool useDerivs**
  flag for use derivatives specification, indicating usage of derivative data (with respect to expansion variables) to enhance the calculation of the stochastic expansion.

- **short refineType**
  refinement type: NO_REFINEMENT, P_REFINEMENT, or H_REFINEMENT

- **short refineControl**
  refinement control: NO_CONTROL, UNIFORM_CONTROL, LOCAL_ADAPTIVE_CONTROL, DIMENSION_ADAPTIVE_CONTROL_SOBOLO, DIMENSION_ADAPTIVE_CONTROL_DECAY, or DIMENSION_ADAPTIVE_CONTROL_GENERALIZED

- **unsigned short softConvLimit**
  number of consecutive iterations within tolerance required to indicate soft convergence

- **RealSymMatrix respCovariance**
  symmetric matrix of analytic response covariance (full response covariance option)

- **RealVector respVariance**
  vector of response variances (diagonal response covariance option)

- **RealVector initialPtU**
  stores the initial variables data in u-space

**Private Member Functions**

- **void reduce_total_sobol_sets** (RealVector &avg_sobol)
  compute average of total Sobol’ indices (from VBD) across the response set for use as an anisotropy indicator

- **void reduce_decay_rate_sets** (RealVector &min_decay)
  compute minimum of spectral coefficient decay rates across the response set for use as an anisotropy indicator

- **void initialize_sets**
  initialization of adaptive refinement using generalized sparse grids

- **Real increment_sets**
  perform an adaptive refinement increment using generalized sparse grids

- **void finalize_sets** (bool converged_within_tol)
  finalization of adaptive refinement using generalized sparse grids

- **void compute_analytic_statistics** (short results_state=FINAL_RESULTS)
  analytic portion of compute_statistics() from post-processing of expansion coefficients

- **void compute_numerical_statistics**
  numerical portion of compute_statistics() from sampling on the expansion

- **void compute_numerical_stat_refinements** (RealVectorArray &imp_sampler_stats, RealRealPairArray &min_max_fns)
  refinements to numerical probability statistics from importanceSampler

- **void compute_covariance**
  calculate the response covariance (diagonal or full matrix)

- **void compute_diagonal_variance**
• void compute_off_diagonal_covariance()  
  calculate respCovariance(i,j) for j<i
• void print_moments(std::ostream &s)  
  print expansion and numerical moments
• void print_covariance(std::ostream &s)  
  print respCovariance
• void print_sobol_indices(std::ostream &s)  
  print global sensitivity indices
• void print_local_sensitivity(std::ostream &s)  
  print local sensitivities evaluated at initialPtU
• void annotated_index_set_results()  
  manage print of results following a generalized index set increment
• void annotated_refinement_results(bool initialize)  
  manage print of results following a refinement increment

Private Attributes
• short ruleNestingOverride  
  user override of default rule nesting: NO_NESTING_OVERRIDE, NESTED, or NON_NESTED
• short ruleGrowthOverride  
  user override of default rule growth: NO_GROWTH_OVERRIDE, RESTRICTED, or UNRESTRICTED
• Iterator expansionSampler  
  Iterator used for sampling on the uSpaceModel to generate approximate probability/reliability/response level statistics. Currently this is an LHS sampling instance, but AIS could also be used.
• Iterator importanceSampler  
  Iterator used to refine the approximate probability estimates generated by the expansionSampler using importance sampling.
• RealMatrix expGradsMeanX  
  derivative of the expansion with respect to the x-space variables evaluated at the means (used as uncertainty importance metrics)
• int maxRefineIterations  
  maximum number of uniform/adaptive refinement iterations (specialization of maxIterations)
• int maxSolverIterations  
  maximum number of regression solver iterations (specialization of maxIterations)
• bool vbdFlag  
  flag indicating the activation of variance-based decomposition for computing Sobol’ indices
• unsigned short vbdOrderLimit  
  limits the order of interactions within the component Sobol’ indices
• Real vbdDropTol  
  tolerance for omitting output of small VBD indices
• short covarianceControl  
  enumeration for controlling response covariance calculation and output: {DEFAULT,DIAGONAL,FULL} COVARIANCE
Additional Inherited Members

13.103.1 Detailed Description

Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC)

The NonDExpansion class provides a base class for methods that use polynomial expansions to approximate the effect of parameter uncertainties on response functions of interest.

13.103.2 Member Function Documentation

const Model & algorithm_space_model() const [inline], [virtual]

default definition that gets redefined in selected derived Minimizers
  Reimplemented from Analyzer.
  References NonDExpansion::uSpaceModel.

Real compute_covariance_metric() [protected], [virtual]

compute 2-norm of change in response covariance
  computes the default refinement metric based on change in respCovariance
  Reimplemented in NonDStochCollocation.
  References NonDExpansion::compute_covariance(), NonDExpansion::covarianceControl, NonDExpansion-::respCovariance, and NonDExpansion::respVariance.
  Referenced by NonDStochCollocation::compute_covariance_metric(), NonDExpansion::increment_sets(), and NonDExpansion::refine_expansion().

Real compute_final_statistics_metric() [protected], [virtual]

compute 2-norm of change in final statistics
  computes a ”goal-oriented” refinement metric employing finalStatistics
  Reimplemented in NonDStochCollocation.
  References NonDExpansion::compute_statistics(), NonD::finalMomentsType, NonD::finalStatistics, Response-::function_values(), Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, Non- D::requestedRelLevels, and NonD::requestedRespLevels.
  Referenced by NonDStochCollocation::compute_final_statistics_metric(), and NonDExpansion::increment_sets().
void compute_statistics ( short results_state = FINAL_RESULTS ) [protected]

calculate analytic and numerical statistics from the expansion

Calculate analytic and numerical statistics from the expansion and log results within final_stats for use in OUU.

References ResultsManager::active(), NonDExpansion::archive_coefficients(), NonDExpansion::archive_moments(), NonDExpansion::compute_analytic_statistics(), NonDExpansion::compute_numerical_statistics(), Model::continuous_variable_labels(), Model::continuous_variables(), NonDExpansion::expansionSampler, NonD::initialize_level_mappings(), NonDExpansion::initialPtU, ResultsManager::insert(), Iterator::is_null(), Iterator::iteratedModel, Model::response_labels(), Iterator::resultsDB, Iterator::resultsNames, Iterator::run_identifier(), NonD::totalLevelRequests, and NonDExpansion::uSpaceModel.

Referenced by NonDExpansion::annotated_index_set_results(), NonDExpansion::annotated_refinement_results(), NonDExpansion::annotated_results(), NonDStochCollocation::compute_final_statistics_metric(), and NonDExpansion::compute_final_statistics_metric().

13.103.3 Member Data Documentation

bool useDerivs [protected]

flag for use_derivatives specification, indicating usage of derivative data (with respect to expansion variables) to enhance the calculation of the stochastic expansion.

This is part of the method specification since the instantiation of the global data fit surrogate is implicit with no user specification. This behavior is distinct from the usage of response derivatives with respect to auxilliary variables (design, epistemic) for computing derivatives of aleatory expansion statistics with respect to these variables.

Referenced by NonDExpansion::compute_expansion(), NonDStochCollocation::initialize_u_space_model(), NonDPolynomialChaos::initialize_u_space_model(), NonDPolynomialChaos::ratio_samples_to_order(), NonDStochCollocation::resolve_inputs(), NonDPolynomialChaos::resolve_inputs(), NonDPolynomialChaos::terms_ratio_to_samples(), and NonDPolynomialChaos::terms_samples_to_ratio().

The documentation for this class was generated from the following files:

- NonDExpansion.hpp
- NonDExpansion.cpp

13.104 NonDGlobalEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

Inheritance diagram for NonDGlobalEvidence:
Public Member Functions

- **NonDGlobalEvidence** (ProblemDescDB &problem_db, Model &model)
  constructor
- **~NonDGlobalEvidence** ()
  destructor
- void **initialize** ()
  perform any required initialization
- void **set_cell_bounds** ()
  set the optimization variable bounds for each cell
- void **get_best_sample** (bool maximize, bool eval_approx)
  determine truthFnStar and approxFnStar
- void **post_process_cell_results** (bool maximize)
  post-process a cell minimization/maximization result
- void **post_process_response_fn_results** ()
  post-process the interval computed for a response function
- void **post_process_final_results** ()
  perform final post-processing

Additional Inherited Members

13.104.1 Detailed Description

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:
13.105 NonDGlobalInterval Class Reference

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDGlobalInterval:

```
NonDGlobalInterval
  |___ NonDInterval
  |   |___ NonD
  |   |   |___ Analyzer
  |   |___ Iterator
  |___ NonDGlobalEvidence
  |___ NonDGlobalSingleInterval

Public Member Functions
- NonDGlobalInterval (ProblemDescDB &problem_db, Model &model) constructor
- NonDGlobalInterval () destructor
- void derived_init_communicators (ParLevLIter pl_iter) derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter) derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter) derived class contributions to freeing the communicators associated with this Iterator instance
- void core_run () Performs an optimization to determine interval bounds for an entire function or interval bounds on a particular statistical estimator.
- const Model & algorithm_space_model () const

Protected Member Functions
- virtual void initialize () perform any required initialization
• virtual void set_cell_bounds ()
  set the optimization variable bounds for each cell
• virtual void get_best_sample (bool maximize, bool eval_approx)
  determine truthFnStar and approxFnStar
• virtual void post_process_cell_results (bool maximize)
  post-process a cell minimization/maximization result
• virtual void post_process_response_fn_results ()
  post-process the interval computed for a response function
• virtual void post_process_final_results ()
  perform final post-processing
• void post_process_run_results (bool maximize)
  post-process an optimization execution: output results, update convergence controls, and update GP approximation
• void evaluate_response_star_truth ()
  evaluate the truth response at the optimal variables solution and update the GP with the new data

Protected Attributes

• Iterator daceIterator
  LHS iterator for constructing initial GP for all response functions.
• Model fHatModel
  GP model of response, one approximation per response function.
• Iterator intervalOptimizer
  optimizer for solving surrogate-based subproblem: NCSU DIRECT optimizer for maximizing expected improvement or mixed EA if discrete variables.
• Model intervalOptModel
  recast model which formulates the surrogate-based optimization subproblem (recasts as design problem; may assimilate mean and variance to enable max(expected improvement))
• Real approxFnStar
  approximate response corresponding to minimum/maximum truth response
• Real truthFnStar
  minimum/maximum truth response function value

Static Private Member Functions

• static void EIF_objective_min (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Expected Improvement Function (EIF) for minimizing the GP
• static void EIF_objective_max (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Expected Improvement Function (EIF) for maximizing the GP
• static void extract_objective (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used to extract the active objective function when optimizing for an interval lower or upper bound (non-EIF formulations). The sense of the optimization is set separately.
CHAPTER 13. CLASS DOCUMENTATION

Private Attributes

- const int seedSpec
  
  the user seed specification (default is 0)
- int numSamples
  
  the number of samples used in the surrogate
- String rngName
  
  name of the random number generator
- bool gpModelFlag
  
  flag indicating use of GP surrogate emulation
- bool eifFlag
  
  flag indicating use of maximized expected improvement for GP iterate selection
- unsigned short improvementConvergeCntr
  
  counter for number of successive iterations that the iteration improvement is less than the convergenceTol
- unsigned short improvementConvergeLimit
  
  counter for number of successive iterations that the iteration improvement is less than the convergenceTol
- Real distanceTol
  
  tolerance for $L_2$ change in optimal solution
- unsigned short distanceConvergeCntr
  
  counter for number of successive iterations that the $L_2$ change in optimal solution is less than the convergenceTol
- unsigned short distanceConvergeLimit
  
  counter for number of successive iterations that the $L_2$ change in optimal solution is less than the convergenceTol
- RealVector prevCVStar
  
  stores previous optimal point for continuous variables; used for assessing convergence
- IntVector prevDIVStar
  
  stores previous optimal point for discrete integer variables; used for assessing convergence
- RealVector prevDRVStar
  
  stores previous optimal point for discrete real variables; used for assessing convergence
- Real prevFinStar
  
  stores previous solution value for assessing convergence
- size_t globalIterCntr
  
  global iteration counter for number of surrogate-based min/max solves
- bool boundConverged
  
  flag indicating convergence of a minimization or maximization cycle
- bool allResponsesPerIter
  
  flag for maximal response extraction (all response values obtained on each function call)
- short dataOrder
  
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec

Static Private Attributes

- static NonDGlobalInterval * nondGIInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
13.106. NONDGLOBALRELIABILITY CLASS REFERENCE

Additional Inherited Members

13.105.1 Detailed Description

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The NonDGlobalInterval class supports global nongradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an “inner loop” aleatory analysis such as intervals on means, variances, or percentile levels. The preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

13.105.2 Member Function Documentation

\texttt{\textbf{const Model & algorithm\_space\_model ( \ ) const}} [inline], [virtual]

default definition that gets redefined in selected derived Minimizers

Reimplemented from Analyzer.
References NonDGlobalInterval::fHatModel.
The documentation for this class was generated from the following files:

- NonDGlobalInterval.hpp
- NonDGlobalInterval.cpp

13.106 NonDGlobalReliability Class Reference

Class for global reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDGlobalReliability:

```
  Iteration
    Analyzer
      NonD
        NonDREligibility
          NonDGlobalReliability
```

Public Member Functions

- \texttt{NonDGlobalReliability (ProblemDescDB &problem\_db, Model &model)}
  \textit{constructor}

- \texttt{\~\textbf{NonDGlobalReliability ( )}}
  \textit{destructor}

- \texttt{bool resize ( )}
reinitializes iterator based on new variable size

- void derived_init_communicators (ParLevLIter pl_iter)
  
  derived class contributions to initializing the communicators associated with this Iterator instance

- void derived_set_communicators (ParLevLIter pl_iter)
  
  derived class contributions to setting the communicators associated with this Iterator instance

- void derived_free_communicators (ParLevLIter pl_iter)
  
  derived class contributions to freeing the communicators associated with this Iterator instance

- void core_run ()

  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)

  print the final iterator results

### Private Member Functions

- void optimize_gaussian_process ()

  construct the GP using EGO/SKO

- void importance_sampling ()

  perform multimodal adaptive importance sampling on the GP

- void get_best_sample ()

  determine current best solution from among sample data for expected improvement function in Performance Measure Approach (PMA)

- Real constraint_penalty (const Real &constraint, const RealVector &c_variables)

  calculate the penalty to be applied to the PMA constraint value

- Real expected_improvement (const RealVector &expected_values, const Variables &recast_vars)

  expected improvement function for the GP

- Real expected_feasibility (const RealVector &expected_values, const Variables &recast_vars)

  expected feasibility function for the GP

### Static Private Member Functions

- static void EIF_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)

  static function used as the objective function in the Expected Improvement (EIF) problem formulation for PMA

- static void EFF_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)

  static function used as the objective function in the Expected Feasibility (EFF) problem formulation for RIA

### Private Attributes

- Real fnStar

  minimum penalized response from among true function evaluations

- short meritFunctionType

  type of merit function used to penalize sample data

- Real lagrangeMult

  Lagrange multiplier for standard Lagrangian merit function.
• Real `augLagrangeMult`
  
  Lagrange multiplier for augmented Lagrangian merit function.
• Real `penaltyParameter`
  
  penalty parameter for augmented Lagrangian merit function
• Real `lastConstraintViolation`
  
  constraint violation at last iteration, used to determine if the current iterate should be accepted (must reduce violation)
• bool `lastIterateAccepted`
  
  flag to determine if last iterate was accepted this controls update of parameters for augmented Lagrangian merit fn
• short `dataOrder`
  
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec

Static Private Attributes

• static `NonDGlobalReliability * nondGlobRelInstance`
  
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

13.106.1 Detailed Description

Class for global reliability methods within DAKOTA/UQ.

The `NonDGlobalReliability` class implements EGO/SKO for global MPP search, which maximizes an expected improvement function derived from Gaussian process models. Once the limit state has been characterized, a multimodal importance sampling approach is used to compute probabilities.

13.106.2 Member Function Documentation

`void core_run()` [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from `Iterator`.

References `NonDGlobalReliability::importance_sampling()`, `NonD::initialize_random_variable_parameters()`, `NonDGlobalReliability::nondGlobRelInstance`, `NonDReliability::numRelAnalyses`, `NonDGlobalReliability::optimize_gaussian_process()`, and `NonD::transform_correlations()`.

`void print_results()` [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in `finalize_run()`.

Reimplemented from `Analyzer`.

References `NonD::cdfFlag`, `NonD::computedGenRelLevels`, `NonD::computedProbLevels`, `NonD::computedRespLevels`, `Iterator::iteratedModel`, `Analyzer::numFunctions`, `NonD::print_densities()`, `Model::response_labels()`, and Dakota::write_precision.

The documentation for this class was generated from the following files:
13.107 NonDGlobalSingleInterval Class Reference

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDGlobalSingleInterval:

```
NonDGlobalSingleInterval
NonDGlobalInterval
NonDInterval
NonD
Analyzer
Iterator
```

### Public Member Functions
- **NonDGlobalSingleInterval (ProblemDescDB &problem_db, Model &model)**
  - constructor
- **~NonDGlobalSingleInterval ()**
  - destructor

### Protected Member Functions
- **void initialize ()**
  - perform any required initialization
- **void post_process_cell_results (bool maximize)**
  - post-process a cell minimization/maximization result
- **void get_best_sample (bool maximize, bool eval_approx)**
  - determine truthFnStar and approxFnStar

### Private Attributes
- **size_t statCntr**
  - counter for finalStatistics
Additional Inherited Members

13.107.1 Detailed Description

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The NonDGlobalSingleInterval class supports global nongradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g., intervals on input = intervals on output), or the intervals may be on statistics of an “inner loop” aleatory analysis such as intervals on means, variances, or percentile levels. The preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

The documentation for this class was generated from the following files:

- NonDGlobalSingleInterval.hpp
- NonDGlobalSingleInterval.cpp

13.108 NonDGPImpSampling Class Reference

Class for the Gaussian Process-based Importance Sampling method.

Inheritance diagram for NonDGPImpSampling:

```
NonDGPImpSampling
    NonDSampling
        NonD
            Analyzer
                Iterator
```

Public Member Functions

- **NonDGPImpSampling (ProblemDescDB &problem_db, Model &model)**
  *standard constructor*

- **~NonDGPImpSampling ()**
  *destructor*

- **bool resize ()**
  *reinitializes iterator based on new variable size*

- **void derived_init_communicators (ParLevLIter pl_iter)**
  *derived class contributions to initializing the communicators associated with this Iterator instance*

- **void derived_set_communicators (ParLevLIter pl_iter)**
  *derived class contributions to setting the communicators associated with this Iterator instance*

- **void derived_free_communicators (ParLevLIter pl_iter)**
derived class contributions to freeing the communicators associated with this Iterator instance

- void core_run ()
  perform the GP importance sampling and return probability of failure
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final statistics
- Real final_probability ()
  returns the probability calculated by the importance sampling

Private Member Functions

- RealVector calcExpIndicator (const int respFnCount, const Real respThresh)
  function to calculate the expected indicator probabilities
- Real calcExpIndPoint (const int respFnCount, const Real respThresh, const RealVector this_mean, const RealVector this_var)
  function to calculate the expected indicator probabilities for one point
- void calcRhoDraw ()
  function to update the rhoDraw data, adding x values and rho draw values
- RealVector drawNewX (int this_k)
  function to pick the next X value to be evaluated by the Iterated model

Private Attributes

- Iterator gpBuild
  LHS iterator for building the initial GP.
- Iterator gpEval
  LHS iterator for sampling on the GP.
- Model gpModel
  GP model of response, one approximation per response function.
- Iterator sampleRhoOne
  LHS iterator for sampling from the rhoOneDistribution.
- int numPtsAdd
  the number of points added to the original set of LHS samples
- int numPtsTotal
  the total number of points
- int numEmulEval
  the number of points evaluated by the GP each iteration
- Real finalProb
  the final calculated probability (p)
- RealVectorArray gpCvars
  Vector to hold the current values of the current sample inputs on the GP.
- RealVectorArray gpMeans
  Vector to hold the current values of the current mean estimates for the sample values on the GP.
- RealVectorArray gpVar
  Vector to hold the current values of the current variance estimates for the sample values on the GP.
• RealVector expIndicator
  Vector to hold the expected indicator values for the current GP samples.
• RealVector rhoDraw
  Vector to hold the rhoDraw values for the current GP samples.
• RealVector normConst
  Vector to hold the normalization constant calculated for each point added.
• RealVector indicator
  IntVector to hold indicator for actual simulation values vs. threshold.
• RealVectorArray xDrawThis
  xDrawThis, appended to locally to hold the X values of emulator points chosen
• RealVector expIndThis
  expIndThis, appended locally to hold the expected indicator
• RealVector rhoDrawThis
  rhoDrawThis, appended locally to hold the rhoDraw density for calculating draws
• RealVector rhoMix
  rhoMix, mixture density
• RealVector rhoOne
  rhoOne, original importance density

Additional Inherited Members

13.108.1 Detailed Description

Class for the Gaussian Process-based Importance Sampling method.

The NonDGPImpSampling implements a method developed by Keith Dalbey that uses a Gaussian process surrogate in the calculation of the importance density. Specifically, the mean and variance of the GP prediction are used to calculate an expected value that a particular point fails, and that is used as part of the computation of the “draw distribution.” The normalization constants and the mixture distribution used are defined in (need to get SAND report).

13.108.2 Constructor & Destructor Documentation

NonDGPImpSampling ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Response::active_set(), Iterator::assign_rep(), Model::assign_rep(), NonD::construct_lhs(), Model::current_response(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), NonDGPImpSampling::gpBuild, NonDGPImpSampling::gpEval, NonDGPImpSampling::gpModel, Model::gradient_type(), Model::hessian_type(), Iterator::iteratedModel, Iterator::maxIterations, NonDGPImpSampling::numEmulEval, NonDGPImpSampling::numPtsAdd, NonDSampling::numSamples, Iterator::outputLevel, Iterator::probDescDB, NonDSampling::randomSeed, ActiveSet::request_values(), NonDSampling::rngName, NonDGPImpSampling::sampleRhoOne, NonDSampling::sampleType, NonDSampling::samplingVarsMode, NonDSampling::statsFlag, NonDSampling::vary_pattern(), and NonDSampling::varyPattern.
13.108.3 Member Function Documentation

```cpp
void core_run() [virtual]
```

perform the GP importance sampling and return probability of failure

Calculate the failure probabilities for specified probability levels using Gaussian process based importance sampling.

Reimplemented from `Iterator`.

References `Model::acv()`, `Iterator::all_responses()`, `Analyzer::all_samples()`, `Iterator::all_samples()`, `Model::append_approximation()`, `Model::approximation_data()`, `Model::approximation_variances()`, `Model::build_approximation()`, `NonDGPImpSampling::calcExpIndicator()`, `NonDGPImpSampling::calcExpIndPoint()`, `NonDGPImpSampling::calcRhoDraw()`, `NonD::cdfFlag`, `NonD::computedProbLevels`, `Model::continuous_lower_bounds()`, `Model::continuous_upper_bounds()`, `Model::continuous_variables()`, `Model::current_response()`, `Model::current_variables()`, `NonDGPImpSampling::drawNewX()`, `Model::evaluate()`, `Model::evaluation_id()`, `NonDGPImpSampling::expIndicator`, `NonDGPImpSampling::expIndThis`, `NonDGPImpSampling::finalProb`, `Response::function_values()`, `NonDGPImpSampling::gpCvars`, `NonDGPImpSampling::gpEval`, `NonDGPImpSampling::gpMeans`, `NonDGPImpSampling::gpModel`, `NonDGPImpSampling::gpVar`, `NonDGPImpSampling::indicator`, `NonD::initialize_level_mappings()`, `Iterator::iteratedModel`, `Iterator::methodPCIter`, `NonD::miPLIndex`, `NonDGPImpSampling::normConst`, `NonDGPImpSampling::numEmulEval`, `Analyzer::numFunctions`, `NonDGPImpSampling::numPtsAdd`, `NonDGPImpSampling::numPtsTotal`, `NonDSampling::numSamples`, `Iterator::outputLevel`, `Model::pop_approximation()`, `NonD::requestedRespLevels`, `NonDGPImpSampling::rhoDraw`, `NonDGPImpSampling::rhoDrawThis`, `NonDGPImpSampling::rhoMix`, `NonDGPImpSampling::rhoOne`, `Iterator::run()`, `NonDGPImpSampling::sampleRhoOne`, and `NonDGPImpSampling::xDrawThis`.

The documentation for this class was generated from the following files:

- `NonDGPImpSampling.hpp`
- `NonDGPImpSampling.cpp`

13.109 NonDGPMSABayesCalibration Class Reference

Generates posterior distribution on model parameters given experiment data.

Inheritance diagram for NonDGPMSABayesCalibration:

```
  Analyzer
    |          |          |          |          |          |
    NonD    NonDCalibration NonDBayesCalibration NonDQUESOBayesCalibration NonDGPMSABayesCalibration
  Iterator
```

The documentation for this class was generated from the following files:

- `NonDGPImpSampling.hpp`
- `NonDGPImpSampling.cpp`
Public Member Functions

- NonDGPMSABayesCalibration (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDGPMSABayesCalibration ()
  destructor

Protected Member Functions

- void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- void calibrate ()
  performs a forward uncertainty propagation by using GPM/SA to generate a posterior distribution on parameters given a set of simulation parameter/response data, a set of experimental data, and additional variables to be specified here.
- void init_queso_solver ()
  specialization to initialize the inverse problem and posterior
- void acquire_simulation_data (RealMatrix &sim_data)
  Populate simulation data (run design of experiments or load build data)
- void overlay_proposal_covariance (QUESO::GslMatrix &full_prop_cov) const
  fill the full proposal covariance, including hyperparameter entries with user-specified or default theta covariance information
- void fill_simulation_data ()
  populate the simulation data, calculating and applying scaling if needed
- void fill_experiment_data ()
  populate the experiment data, applying scaling if needed
- void overlay_initial_params (QUESO::GslVector &full_param_initials)
  overlay the Dakota user’s initial parameters on the full GPMSA vector of parameters
- void cache_acceptance_chain ()
  retrieve the chain from QUESO and populate acceptanceChain / acceptedFnVals
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results

Protected Attributes

- int buildSamples
  number of samples of the simulation to construct the GP
- String approxImportFile
  name of file from which to import build points to build GP
- unsigned short approxImportFormat
  build data import tabular format
• bool approxImportActiveOnly
  import active variables only
• unsigned int userConfigVars
  number of user-specified configuration (scenario) vars
• unsigned int gpmsaConfigVars
  number of config vars presented to GPMSA (minimum 1)
• bool gpmsaNormalize
  whether to apply GPMSA-internal variable and data normalization
• boost::shared_ptr
  QUESO::VectorSpace
  QUESO::GslVector,
  QUESO::GslMatrix > > configSpace
  vector space defining the scenario (configuration) variables
• boost::shared_ptr
  QUESO::VectorSpace
  QUESO::GslVector,
  QUESO::GslMatrix > > nEtaSpace
  vector space defining the output (response) space for the simulations
• boost::shared_ptr
  QUESO::VectorSpace
  QUESO::GslVector,
  QUESO::GslMatrix > > experimentSpace
  vector space defining the output (response) space for the experiments
• boost::shared_ptr
  QUESO::GPMSAOptions > gpmsaOptions
  Configuration options for the GPMSA solver.
• boost::shared_ptr
  QUESO::GPMSAFactory
  QUESO::GslVector,
  QUESO::GslMatrix > > gpmsaFactory
  core factory that manages a GP-based likelihood

Private Attributes

• Iterator lhsIter
  LHS iterator for generating samples for GP.

Static Private Attributes

• static NonDGPSMABayesCalibration * nonDGPSMAModule
  Pointer to current class instance for use in static callback functions.
### Additional Inherited Members

#### 13.109.1 Detailed Description

Generates posterior distribution on model parameters given experiment data.

This class provides a wrapper for the functionality provided in the Los Alamos National Laboratory code called GPM/SA (Gaussian Process Models for Simulation Analysis). Although this is a code that provides input/output mapping, it DOES NOT provide the mapping that we usually think of in the NonDeterministic class hierarchy in DAKOTA, where uncertainty in parameter inputs are mapped to uncertainty in simulation responses. Instead, this class takes a pre-existing set of simulation data as well as experimental data, and maps priors on input parameters to posterior distributions on those input parameters, according to a likelihood function. The goal of the MCMC sampling is to produce posterior values of parameter estimates which will produce simulation response values that "match well" to the experimental data. The MCMC is an integral part of the calibration. The data structures in GPM/SA are fairly detailed and nested. Part of this prototyping exercise is to determine what data structures need to be specified and initialized in DAKOTA and sent to GPM/SA, and what data structures will be returned.

#### 13.109.2 Constructor & Destructor Documentation

**NonDGPMsABayesCalibration** ( ProblemDescDB & _problem_db_, Model & _model_ )

*constructor*

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDGPMsABayesCalibration::approxImportActiveOnly, NonDGPMsABayesCalibration::approxImportFile, Iterator::assign_rep(), NonDGPMsABayesCalibration::buildSamples, Model::current_response(), NonDBayesCalibration::emulatorType, NonDCalibration::expData, ProblemDescDB::get_string(), NonDGPMsABayesCalibration::lhsIter, NonDBayesCalibration::mcmcModel, ExperimentData::num_experiments(), SharedResponseData::num_field_response_groups(), Iterator::outputLevel, Iterator::probDescDB, NonDBayesCalibration::randomSeed, Response::shared_data(), and NonDGPMsABayesCalibration::userConfigVars.

#### 13.109.3 Member Function Documentation

**void calibrate()** [protected], [virtual]

performs a forward uncertainty propagation by using GPM/SA to generate a posterior distribution on parameters given a set of simulation parameter/response data, a set of experimental data, and additional variables to be specified here.

Perform the uncertainty quantification

Implements NonDBayesCalibration.

References NonDQUESOBayesCalibration::advancedOptionsFile, ExperimentData::all_data(), NonDGPMsABayesCalibration::buildSamples, NonDGPMsABayesCalibration::cache_acceptance_chain(), NonDQUESOBayesCalibration::callIpMhOptionsValues, NonDBayesCalibration::compute_statistics(), NonDGPMsABayesCalibration::configSpace, NonDCalibration::expData, NonDGPMsABayesCalibration::experimentSpace, NonDGPMsABayesCalibration::fill_experiment_data(), NonDGPMsABayesCalibration::fill_simulation_data(), NonDGPMsABayesCalibration::gpmsaConfigVars, NonDGPMsABayesCalibration::gpmsaFactory, NonDGPMsABayesCalibration::gpmsaNormalize, NonDGPMsABayesCalibration::gpmsaOptions, NonDQUESOBayesCalibration::init_parameter_domain(), NonDGPMsABayesCalibration::init_queso_solver(), NonDBayesCalibration::initialize_model(), NonDQUESOBayesCalibration::mcmcType, NonDGPMsABayesCalibration::nEtaSpace, NonDGPMsABayesCalibration::nonDGPMsAInstance, NonDQUESOBayesCalibration::nonDQUESOInstance, ExperimentData::num_experiments(),
Analyzer::numContinuousVars, Analyzer::numFunctions, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, NonDGPMGMSABayesCalibration::overlay_initial_params(), NonDGPMGMSABayesCalibration::overlay_proposal_covariance(), NonDQUESOBayesCalibration::paramSpace, NonDQUESOBayesCalibration::quesoEnv, and NonDBayesCalibration::standardizedSpace.

```cpp
void fill_simulation_data( ) [protected]
```
populate the simulation data, calculating and applying scaling if needed

- simulation data, one row per simulation build sample, columns for calibration variables, configuration variables, function values (duplicates storage, but unifies import vs. DOE cases)

  References NonDGPMGMSABayesCalibration::acquire_simulation_data(), NonDGPMGMSABayesCalibration::buildSamples, NonDGPMGMSABayesCalibration::configSpace, NonDGPMGMSABayesCalibration::gpmsaConfigVars, NonDGPMGMSABayesCalibration::gpmsaFactory, NonDGPMGMSABayesCalibration::nEtaSpace, Analyzer::numContinuousVars, NonDQUESOBayesCalibration::paramSpace, and NonDGPMGMSABayesCalibration::userConfigVars.

  Referenced by NonDGPMGMSABayesCalibration::calibrate().

```cpp
void cache_acceptance_chain( ) [protected]
```
retrieve the chain from QUESO and populate acceptanceChain / acceptedFnVals

  This is a subset of the base class retrieval, but we can’t do the fn value lookups. Eventually should be able to retrieve them from GPMSA.

  References NonDBayesCalibration::acceptanceChain, NonDBayesCalibration::acceptedFnVals, NonDBayesCalibration::chainCycles, NonDBayesCalibration::chainSamples, NonDQUESOBayesCalibration::copy_gsl_partial(), NonD::natafTransform, Analyzer::numContinuousVars, Analyzer::numFunctions, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, and NonDBayesCalibration::standardizedSpace.

  Referenced by NonDGPMGMSABayesCalibration::calibrate().

```cpp
void print_results( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
```
print the final iterator results

  This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

  Reimplemented from NonDBayesCalibration.

  References NonDBayesCalibration::print_results().

  The documentation for this class was generated from the following files:

- NonDGPMGMSABayesCalibration.hpp
- NonDGPMGMSABayesCalibration.cpp

### 13.110 NonDIntegration Class Reference

Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.

Inheritance diagram for NonDIntegration:
Public Member Functions

- virtual void `initialize_grid` (const std::vector<Pecos::BasisPolynomial> &poly_basis)=0
  initialize integration grid by drawing from polynomial basis settings
- virtual void `increment_grid` ()=0
  increment SSG level/TPQ order
- virtual void `increment_grid_preference` (const RealVector &dim_pref)
  increment SSG level/TPQ order and update anisotropy
- virtual void `increment_grid_weights` (const RealVector &aniso_wts)
  increment SSG level/TPQ order and update anisotropy
- const Pecos::IntegrationDriver & `driver` () const
  return numIntDriver
- bool `resize` ()
  reinitializes iterator based on new variable size

Static Public Member Functions

- static void `dimension_preference_to_anisotropic_order` (unsigned short scalar_order_spec, const RealVector &dim pref spec, size_t num_v, UShortArray &aniso_order)
  convert scalar order spec and vector dim pref spec to vector aniso order
- static void `anisotropic_order_to_dimension_preference` (const UShortArray &aniso_order, unsigned short &scalar_order, RealVector &dim pref)
  convert vector aniso order to scalar_order and vector dim pref

Protected Member Functions

- `NonDIntegration` (ProblemDescDB &problem db, Model &model)
  constructor
- `NonDIntegration` (unsigned short method_name, Model &model)
  alternate constructor for instantiations "on the fly"
- `NonDIntegration` (unsigned short method_name, Model &model, const RealVector &dim pref)
  alternate constructor for instantiations "on the fly"
• NonDIntegration ()
  destructor
• void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
• void check_variables (const std::vector<Pecos::RandomVariable> &x_ran_vars)
  verify self-consistency of variables data
• void print_points_weights (const String &tabular_name)
  output integration points and weights to a tabular file

Protected Attributes
• Pecos::IntegrationDriver numIntDriver
  Pecos utility class for managing interface to tensor-product grids and VPISparseGrid utilities for Smolyak sparse
  grids and cubature.
• size_t numIntegrations
  counter for number of integration executions for this object
• RealVector dimPrefSpec
  the user specification for anisotropic dimension preference

Additional Inherited Members

13.110.1 Detailed Description
Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of ex-
pectation integrals.
This class provides a base class for shared code among NonDQuadrature and NonDSparseGrid.

13.110.2 Constructor & Destructor Documentation
NonDIntegration ( ProblemDescDB & problem_db, Model & model ) [protected]
constructor
  This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes
  has been called and probDescDB can be queried for settings from the method specification. It is not currently
  used, as there are not yet separate nond_quadature/nond_sparse_grid method specifications.
  References Dakota::abort_handler(), NonD::initialize_final_statistics(), NonD::initialize_random_variable_correlations(),
  NonD::initialize_random_variable_transformation(), NonD::initialize_random_variable_types(), Analyzer::numDiscrete-
  IntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, and NonD::verify_correlation_support().

NonDIntegration ( unsigned short method_name, Model & model ) [protected]
alternate constructor for instantiations ”on the fly”
  This alternate constructor is used for on-the-fly generation and evaluation of numerical integration points.

NonDIntegration ( unsigned short method_name, Model & model, const RealVector & dim pref ) [protected]
alternate constructor for instantiations ”on the fly”
  This alternate constructor is used for on-the-fly generation and evaluation of numerical integration points.
13.111. NONDINTERVAL CLASS REFERENCE

13.110.3 Member Function Documentation

```cpp
void dimension_preference_to_anisotropic_order ( unsigned short scalar_order_spec, const RealVector & dim_pref_spec, size_t num_v, UShortArray & aniso_order ) [static]
```

convert scalar_order_spec and vector dim_pref_spec to vector aniso_order

Converts a scalar order specification and a vector anisotropic dimension preference into an anisotropic order vector. It is used for initialization and does not enforce a reference lower bound (see also NonDQuadrature::*update_anisotropic_order()).

Referenced by NonDPolynomialChaos::config_expansion_orders(), NonDMultilevelPolynomialChaos::incrementSpecification_sequence(), and NonDQuadrature::initialize_dimension_quadrature_order().

```cpp
void anisotropic_order_to_dimension_preference ( const UShortArray & aniso_order, unsigned short & scalar_order, RealVector & dim_pref ) [static]
```

convert vector aniso_order to scalar_order and vector dim_pref

Converts a vector anisotropic order into a scalar order and vector anisotropic dimension preference.

Referenced by NonDPolynomialChaos::config_regression().

```cpp
void core_run ( ) [protected], [virtual]
```

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References NonDIntegration::check_variables(), Analyzer::evaluate_parameter_sets(), Analyzer::getParameterSets(), Iterator::iteratedModel, NonD:natafTransform, and NonDIntegration::numIntegrations.

```cpp
void check_variables ( const std::vector<Pecos::RandomVariable> & x_ran_vars ) [protected]
```

verify self-consistency of variables data

Virtual function called from probDescDB-based constructors and from NonDIntegration::core_run()

References Dakota::abort_handler(), NonD::numContAleatUncVars, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContIntervalVars, Analyzer::numContinuousVars, and NonD::numContStateVars.

Referenced by NonDIntegration::core_run(), NonDCubature::NonDCubature(), NonDQuadrature::NonDQuadrature(), and NonDSparseGrid::NonDSparseGrid().

The documentation for this class was generated from the following files:

- NonDIntegration.hpp
- NonDIntegration.cpp

13.111 NonDInterval Class Reference

Base class for interval-based methods within DAKOTA/UQ.

Inheritance diagram for NonDInterval:
Public Member Functions

- `NonDInterval (ProblemDescDB &problem_db, Model &model)`
  * constructor
- `~NonDInterval ()`
  * destructor
- `void print_results (std::ostream &s, short results_state=FINAL_RESULTS)`
  * print the cumulative distribution functions for belief and plausibility
- `bool resize ()`
  * reinitializes iterator based on new variable size

Protected Member Functions

- `void initialize_final_statistics ()`
  * initialize finalStatistics for belief/plausibility results sets
- `void compute_evidence_statistics ()`
  * method for computing belief and plausibility values for response levels or vice-versa
- `void calculate_cells_and_bpas ()`
  * computes the interval combinations (cells) and their bpas replaces CBPIIC_F77 from wrapper calculate_basic_prob_intervals()
- `void calculate_cbf_cpf (bool complementary=true)`
  * function to compute (complementary) distribution functions on belief and plausibility replaces CCBFPF_F77 from wrapper calculate_cum_belief_plaus()

Protected Attributes

- `bool singleIntervalFlag`
  * flag for SingleInterval derived class
- `RealVectorArray ccBelFn`
  * Storage array to hold CCBF values.
- `RealVectorArray ccPlausFn`
  * Storage array to hold CCPF values.
- `RealVectorArray ccBelVal`
  * Storage array to hold CCB response values.
- `RealVectorArray ccPlausVal`
  * Storage array to hold CCP response values.
13.111. NONDINTERVAL CLASS REFERENCE

- RealVectorArray cellContLowerBounds
  Storage array to hold cell lower bounds for continuous variables.
- RealVectorArray cellContUpperBounds
  Storage array to hold cell upper bounds for continuous variables.
- IntVectorArray cellIntRangeLowerBounds
  Storage array to hold cell lower bounds for discrete int range variables.
- IntVectorArray cellIntRangeUpperBounds
  Storage array to hold cell upper bounds for discrete int range variables.
- IntVectorArray cellIntSetBounds
  Storage array to hold cell values for discrete integer set variables.
- IntVectorArray cellRealSetBounds
  Storage array to hold cell value for discrete real set variables.
- RealVectorArray cellFnLowerBounds
  Storage array to hold cell min.
- RealVectorArray cellFnUpperBounds
  Storage array to hold cell max.
- RealVector cellBPA
  Storage array to hold cell bpa.
- size_t respFnCntr
  response function counter
- size_t cellCntr
  cell counter
- size_t numCells
  total number of interval combinations

Additional Inherited Members

13.111.1 Detailed Description

Base class for interval-based methods within DAKOTA/UQ.

The NonDInterval class implements the propagation of epistemic uncertainty using either pure interval propagation or Dempster-Shafer theory of evidence. In the latter approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:

- NonDInterval.hpp
- NonDInterval.cpp
13.112 NonDLHSEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

Inheritance diagram for NonDLHSEvidence:

```
NonDLHSEvidence
   |             |
   |             |
   |             |
   |             |
   |             |
   |             |
   |             |
   |             |
   |             |
   |             |
   |             |
   |             |
```

Public Member Functions

- **NonDLHSEvidence** (ProblemDescDB &problem_db, Model &model)
  
  *constructor*

- **~NonDLHSEvidence** ()
  
  *destructor*

- **void initialize ()**
  
  *perform any required initialization*

- **void post_process_samples ()**
  
  *post-process the output from executing lhsSampler*

Additional Inherited Members

13.112.1 Detailed Description

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:

- NonDLHSEvidence.hpp
- NonDLHSEvidence.cpp
13.113 NonDLHSInterval Class Reference

Class for the LHS-based interval methods within DAKOTA/UQ.

Inheritance diagram for NonDLHSInterval:

![Inheritance Diagram]

**Public Member Functions**

- `NonDLHSInterval (ProblemDescDB &problem_db, Model &model)`
  
  constructor

- `~NonDLHSInterval ()`
  
  destructor

- `void derived_init_communicators (ParLevLIter pl_iter)`
  
  derived class contributions to initializing the communicators associated with this Iterator instance

- `void derived_set_communicators (ParLevLIter pl_iter)`
  
  derived class contributions to setting the communicators associated with this Iterator instance

- `void derived_free_communicators (ParLevLIter pl_iter)`
  
  derived class contributions to freeing the communicators associated with this Iterator instance

- `void core_run ()`
  
  performs an epistemic uncertainty propagation using LHS samples

**Protected Member Functions**

- `virtual void initialize ()`
  
  perform any required initialization

- `virtual void post_process_samples ()=0`
  
  post-process the output from executing lhsSampler
CHAPTER 13. CLASS DOCUMENTATION

Protected Attributes

• Iterator lhsSampler
  the LHS sampler instance
• const int seedSpec
  the user seed specification (default is 0)
• int numSamples
  the number of samples used
• String rngName
  name of the random number generator

Additional Inherited Members

13.113.1 Detailed Description

Class for the LHS-based interval methods within DAKOTA/UQ.
The NonDLHSInterval class implements the propagation of epistemic uncertainty using LHS-based methods.
The documentation for this class was generated from the following files:

• NonDLHSInterval.hpp
• NonDLHSInterval.cpp

13.114 NonDLHSSampling Class Reference

Performs LHS and Monte Carlo sampling for uncertainty quantification.
Inheritance diagram for NonDLHSSampling:

```
NonDLHSSampling
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

Public Member Functions

• NonDLHSSampling (ProblemDescDB &problem_db, Model &model)
  standard constructor
• NonDLHSSampling (Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern=true, short sampling_vars_mode=ACTIVE)
  alternate constructor for sample generation and evaluation "on the fly"
• **NonDLHSSampling** (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  *alternate constructor for uniform sample generation "on the fly"

• **NonDLHSSampling** (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &means, const RealVector &std_devs, const RealVector &lower_bnds, const RealVector &upper_bnds, RealSymMatrix &correl)

  *alternate constructor for sample generation of correlated normals "on the fly"

• ~**NonDLHSSampling** ()

  *destructor

**Protected Member Functions**

• void **sampling_increment** ()
  
  *increment to next in sequence of refinement samples

• void **pre_run** ()
  
  *generate LHS samples in non-VBD cases

• void **core_run** ()
  
  *perform the evaluate parameter sets portion of run

• void **post_run** (std::ostream &s)
  
  *generate statistics for LHS runs in non-VBD cases

• void **post_input** ()
  
  *read tabular data for post-run mode

• void **update_final_statistics** ()
  
  *update finalStatistics and (if MC sampling) finalStatErrors

• void **compute_pca** (std::ostream &s)
  
  *compute a principal components analysis on the sample set

• void **print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  
  *print the final statistics

• void **d_optimal_parameter_set** (int previous_samples, int new_samples, RealMatrix &full_samples)
  
  *generate a d-optimal parameter set, leaving the first previous_samples columns intact and adding new_samples new columns following them

• void **initial_increm_lhs_set** (int new_samples, RealMatrix &full_samples, IntMatrix &full_ranks)
  
  *Populate the first new_samples columns of allSamples with an LHS design and update the stored ranks.

• void **increm_lhs_parameter_set** (int previous_samples, int new_samples, RealMatrix &full_samples, IntMatrix &full_ranks)
  
  *generate a new batch that is Latin w.r.t. the previous samples

• void **store_ranks** (const RealMatrix &sample_values, IntMatrix &sample_ranks)
  
  *store the ranks of the last generated sample for continuous (based on sampleRanks) and calculate/store discrete ranks

• void **store_ranks** (IntMatrix &full_ranks)
  
  *store the combined ranks from sampleRanks to leading submatrix local cached ranks matrix

• void **combine_discrete_ranks** (const RealMatrix &initial_values, const RealMatrix &increm_values)
  
  *merge the discrete ranks into a submatrix of sampleRanks

• void **print_header_and_statistics** (std::ostream &s, const int &num_samples)
  
  *Print a header and summary statistics.
Static Protected Member Functions

- static bool rank_sort (const int &x, const int &y)
  sort algorithm to compute ranks for rank correlations

Private Attributes

- size_t numResponseFunctions
  number of response functions; used to distinguish NonD from opt/NLS usage
- IntVector refineSamples
  list of refinement sample batch sizes
- bool dOptimal
  whether to generate d-optimal point sets
- size_t numCandidateDesigns
  number of candidate designs to generate for classical D-optimal designs
- Real oversampleRatio
  oversampling ratio for Leja D-optimal candidate set generation
- bool varBasedDecompFlag
  flags computation of variance-based decomposition indices
- bool pcaFlag
  flag to specify the calculation of principal components
- Real percentVarianceExplained
  Threshold to keep number of principal components that explain this much variance.

Static Private Attributes

- static RealArray rawData
  static data used by static rank_sort() fn

Additional Inherited Members

13.114.1 Detailed Description

Performs LHS and Monte Carlo sampling for uncertainty quantification.

The Latin Hypercube Sampling (LHS) package from Sandia Albuquerque’s Risk and Reliability organization provides comprehensive capabilities for Monte Carlo and Latin Hypercube sampling within a broad array of user-specified probabilistic parameter distributions. It enforces user-specified rank correlations through use of a mixing routine. The NonDLHSSampling class provides a C++ wrapper for the LHS library and is used for performing forward propagations of parameter uncertainties into response statistics.

Batch generation options, including D-Optimal and incremental LHS are provided.

The incremental LHS sampling capability allows one to supplement an initial sample of size n to size 2n while maintaining the correct stratification of the 2n samples and also maintaining the specified correlation structure. The incremental version of LHS will return a sample of size n, which when combined with the original sample of size n, allows one to double the size of the sample.
13.114.2 Constructor & Destructor Documentation

**NonDLHSSampling ( ProblemDescDB & problem_db, Model & model )**

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDLHSSampling::dOptimal, Model::num_primary_fns(), NonD::num-BinomialVars, NonDLHSSampling::numCandidateDesigns, NonD::numDesignVars, NonD::numEpistemicUnc-Vars, NonD::numGeometricVars, NonD::numHistogramPtIntVars, NonD::numHistogramPtRealVars, NonD::num-HistogramPtStringVars, NonD::numHyperGeomVars, NonD::numNegBinomialVars, NonD::numPoissonVars, NonDLHSSampling::numResponseFunctions, NonD::numStateVars, Iterator::outputLevel, NonDLHSSampling::oversample-Ratio, Model::primary_fn_type(), NonDLHSSampling::refineSamples, and NonDSampling::sampleType.

**NonDLHSSampling ( Model & model, unsigned short sample_type, int samples, int seed, const String & rng, bool vary_pattern = true, short sampling_vars_mode = ACTIVE )**

alternate constructor for sample generation and evaluation "on the fly"

This alternate constructor is used for generation and evaluation of Model-based sample sets. A set_db_list_nodes has not been performed so required data must be passed through the constructor. It’s purpose is to avoid the need for a separate LHS specification within methods that use LHS sampling.

**NonDLHSSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds )**

alternate constructor for uniform sample generation "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets. It is not a letter-envelope instantiation and a set_db_list_nodes has not been performed. It is called with all needed data passed through the constructor and is designed to allow more flexibility in variables set definition (i.e., relax connection to a variables specification and allow sampling over parameter sets such as multiobjective weights). In this case, a Model is not used and the object must only be used for sample generation (no evaluation).

References NonDSampling::get_parameter_sets().

**NonDLHSSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & means, const RealVector & std_devs, const RealVector & lower_bnds, const RealVector & upper_bnds, RealSymMatrix & correl )**

alternate constructor for sample generation of correlated normals "on the fly"

This alternate constructor is used to generate correlated normal sample sets. It is not a letter-envelope instantiation and a set_db_list_nodes has not been performed. It is called with all needed data passed through the constructor. In this case, a Model is not used and the object must only be used for sample generation (no evaluation).

References NonDSampling::get_parameter_sets().

13.114.3 Member Function Documentation

**void core_run ( ) [protected], [virtual]**

perform the evaluate parameter sets portion of run

Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.
Reimplemented from `Iterator`.
References `NonDSampling::allDataFlag`, `Analyzer::evaluate_parameter_sets()`, `Iterator::iteratedModel`, `NonDLHSSampling::numResponseFunctions`, and `NonDSampling::statsFlag`.

```cpp
void d_optimal_parameter_set ( int previous_samples, int new_samples, RealMatrix & full_samples )
[protected]
```

generate a d-optimal parameter set, leaving the first previous_samples columns intact and adding new_samples new columns following them.

For now, when this function is called, numSamples is the number of new samples to generate.
References `Model::aleatory_distribution_parameters()`, `Dakota::det_AtransA()`, `NonDSampling::get_parameter_sets()`, `Iterator::iteratedModel`, `NonDSampling::mode_counts()`, `NonD::natafTransform`, `NonDLHSSampling::numCandidateDesigns`, `Iterator::outputLevel`, `NonDLHSSampling::oversampleRatio`, and `NonDSampling::transform_samples()`.

Referenced by `NonDLHSSampling::pre_run()`.
The documentation for this class was generated from the following files:

- NonDLHSSampling.hpp
- NonDLHSSampling.cpp

### 13.115 NonDLHSSingleInterval Class Reference

Class for pure interval propagation using LHS.

Inheritance diagram for `NonDLHSSingleInterval`:

```plaintext
  NonDLHSSingleInterval
   ` NonDLHSInterval
    ` NonDInterval
     ` NonD
      ` Iterator
         ` ` Analyzer
```

Public Member Functions

- **NonDLHSSingleInterval (ProblemDescDB &problem_db, Model &model)**
  
  *constructor*
- **~NonDLHSSingleInterval**
  
  *destructor*
Protected Member Functions

- void initialize ()
  
  perform any required initialization

- void post_process_samples ()
  
  post-process the output from executing lhsSampler

Private Attributes

- size_t statCntr
  
  counter for finalStatistics

Additional Inherited Members

13.115.1 Detailed Description

Class for pure interval propagation using LHS.
The NonDSingleInterval class implements the propagation of epistemic uncertainty using ...
The documentation for this class was generated from the following files:

- NonDLHSSingleInterval.hpp
- NonDLHSSingleInterval.cpp

13.116 NonDLocalEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.
Inheritance diagram for NonDLocalEvidence:

```
NonDLocalEvidence
NonDLocalInterval
NonDInterval
NonD
Analyzer
Iterator
```

Public Member Functions

- NonDLocalEvidence (ProblemDescDB &problem_db, Model &model)
  
  constructor

- ~NonDLocalEvidence ()
  
  destructor
Protected Member Functions

- void initialize ()
  *perform any required initialization*
- void set_cell_bounds ()
  *set the optimization variable bounds for each cell*
- void truncate_to_cell_bounds (RealVector &initial_pt)
  *truncate initial_pt to respect current cell lower/upper bounds*
- void post_process_cell_results (bool maximize)
  *post-process a cell minimization/maximization result*
- void post_process_response_fn_results ()
  *post-process the interval computed for a response function*
- void post_process_final_results ()
  *perform final post-processing*

Additional Inherited Members

13.116.1 Detailed Description

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:

- NonDLocalEvidence.hpp
- NonDLocalEvidence.cpp

13.117 NonDLocalInterval Class Reference

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDLocalInterval:
Public Member Functions

- `NonDLocalInterval (ProblemDescDB &problem_db, Model &model)`  
  constructor

- `~NonDLocalInterval ()`  
  destructor

- `void derived_init_communicators (ParLevLIter pl_iter)`  
  derived class contributions to initializing the communicators associated with this Iterator instance

- `void derived_set_communicators (ParLevLIter pl_iter)`  
  derived class contributions to setting the communicators associated with this Iterator instance

- `void derived_free_communicators (ParLevLIter pl_iter)`  
  derived class contributions to freeing the communicators associated with this Iterator instance

- `void core_run ()`  
  Performs a gradient-based optimization to determine interval bounds for an entire function or interval bounds on a particular statistical estimator.

- `unsigned short uses_method () const`  
  return name of active optimizer method

- `void method_recourse ()`  
  perform an MPP optimizer method switch due to a detected conflict

Protected Member Functions

- `virtual void initialize ()`  
  perform any required initialization

- `virtual void set_cell_bounds ()`  
  set the optimization variable bounds for each cell

- `virtual void truncate_to_cell_bounds (RealVector &initial_pt)`  
  truncate initial pt to respect current cell lower/upper bounds

- `virtual void post_process_cell_results (bool maximize)`  
  post-process a cell minimization/maximization result
• virtual void post_process_response_fn_results ()
  post-process the interval computed for a response function
• virtual void post_process_final_results ()
  perform final post-processing

Protected Attributes

• Iterator minMaxOptimizer
  local gradient-based optimizer
• Model minMaxModel
  recast model which extracts the active objective function

Static Private Member Functions

• static void extract_objective (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used to extract the active objective function when optimizing for an interval lower or upper bound

Private Attributes

• bool npsolFlag
  flag representing the gradient-based optimization algorithm selection (NPSOL SQP or OPT++ NIP)

Static Private Attributes

• static NonDLocalInterval * nondLInstance
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

13.117.1 Detailed Description

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The NonDLocalInterval class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g., intervals on input = intervals on output), or the intervals may be on statistics of an ”inner loop” aleatory analysis such as intervals on means, variances, or percentile levels.

The documentation for this class was generated from the following files:

• NonDLocalInterval.hpp
• NonDLocalInterval.cpp
**13.118 NonDLocalReliability Class Reference**

Class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDLocalReliability:

```
    Iterator
     |    
    ---|---
     |    
    Analyzer
     |    
    ---|---
     |    
    NonD
     |    
    ---|---
     |    
    NonDReliability
     |    
    ---|---
     |    
NonDLocalReliability
```

**Public Member Functions**

- **NonDLocalReliability** (ProblemDescDB &problem_db, Model &model)
  
  *constructor*

- **~NonDLocalReliability** ()
  
  *destructor*

- **void derived_init_communicators** (ParLevLIter pl_iter)
  
  derived class contributions to initializing the communicators associated with this Iterator instance

- **void derived_set_communicators** (ParLevLIter pl_iter)
  
  derived class contributions to setting the communicators associated with this Iterator instance

- **void derived_free_communicators** (ParLevLIter pl_iter)
  
  derived class contributions to freeing the communicators associated with this Iterator instance

- **void core_run** ()
  
  performs an uncertainty propagation using analytical reliability methods which solve constrained optimization problems to obtain approximations of the cumulative distribution function of response

- **void print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  
  print the approximate mean, standard deviation, and importance factors when using the mean value method or the CDF/CCDF information when using MPP-search-based reliability methods

- **unsigned short uses_method** () const
  
  return name of active MPP optimizer

- **void method_recourse** ()
  
  perform an MPP optimizer method switch due to a detected conflict

**Private Member Functions**

- **void initial_taylor_series** ()
  
  convenience function for performing the initial limit state Taylor-series approximation

- **void mean_value** ()
convenience function for encapsulating the simple Mean Value computation of approximate statistics and importance factors

- **void mpp_search ()**
  
  convenience function for encapsulating the reliability methods that employ a search for the most probable point (AMV, AMV+, FORM, SORM)

- **void initialize_class_data ()**
  
  convenience function for initializing class scope arrays

- **void initialize_level_data ()**
  
  convenience function for initializing/warm starting MPP search data for each response function prior to level 0

- **void initialize_mpp_search_data ()**
  
  convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function

- **void mpp_search_data (const Variables &vars_star, const Response &resp_star)**
  
  convenience function for updating MPP search data for each z/p/beta level for each response function

- **void update_mpp_search_data ()**
  
  convenience function for updating MPP search data for each z/p/beta level for each response function

- **void update_level_data ()**
  
  convenience function for updating z/p/beta level data and final statistics following MPP convergence

- **void update_pma_maximize (const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)**
  
  update pmaMaximizeG from prescribed probabilities or prescribed generalized reliabilities by inverting second-order integrations

- **void update_limit_state_surrogate ()**
  
  convenience function for passing the latest variables/response data to the data fit embedded within uSpaceModel

- **void assign_mean_data ()**
  
  update mostProbPointX/U, computedRespLevel, fnGradX/U, and fnHessX/U from ranVarMeansX/U, fnValsMeanX, fnGradsMeanX, and fnHessiansMeanX

- **void dg_ds_eval (const RealVector &x_vars, const RealVector &fn_grad_x, RealVector &final_stat_grad)**
  
  convenience function for evaluating dg/ds

- **Real dp2_dbeta_factor (Real beta, bool cdf_flag)**
  
  compute factor for derivative of second-order probability with respect to reliability index (from differentiating BREITUNG or HOHENRACK expressions)

- **Real signed_norm (const RealVector &mpp_u, const RealVector &fn_grad_u, bool cdf_flag)**
  
  convert norm of mpp_u (u-space solution) to a signed reliability index

- **Real signed_norm (Real norm_mpp_u)**
  
  convert norm of u-space vector to a signed reliability index

- **Real signed_norm (Real norm_mpp_u, const RealVector &mpp_u, const RealVector &fn_grad_u, bool cdf_flag)**
  
  shared helper function

- **Real probability (Real beta)**
  
  Convert reliability to probability using a first-order integration.

- **Real probability (bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)**
  
  Convert computed reliability to probability using either a first-order or second-order integration.

- **Real probability (Real beta, bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)**
  
  Convert provided reliability to probability using either a first-order or second-order integration.
• Real reliability (Real p)
  Convert probability to reliability using the inverse of a first-order integration.

• Real reliability (Real p, bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)
  Convert probability to reliability using the inverse of a first-order or second-order integration.

• bool reliability_residual (const Real &p, const Real &beta, const RealVector &kappa, Real &res)
  compute the residual for inversion of second-order probability corrections using Newton’s method (called by reliability(p))

• Real reliability_residual_derivative (const Real &p, const Real &beta, const RealVector &kappa)
  compute the residual derivative for inversion of second-order probability corrections using Newton’s method (called by reliability(p))

• void principal_curvatures (const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u, RealVector &kappa_u)
  Compute the kappaU vector of principal curvatures from fnHessU.

• void scale_curvature (Real beta, bool cdf_flag, const RealVector &kappa, RealVector &scaled_kappa)
  scale copy of principal curvatures by -1 if needed; else take a view

Static Private Member Functions

• static void RIA_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of (norm u)^2.

• static void RIA_constraint_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of G(u) = response level.

• static void PMA_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of G(u).

• static void PMA_constraint_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the constraint function in the first-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of (norm u)^2 = (beta-bar)^2.

• static void PMA2_constraint_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the constraint function in the second-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of beta* = beta-bar.

• static void PMA2_set_mapping (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  static function used to augment the sub-model ASV requests for second-order PMA
Private Attributes

- Real computedRespLevel
  output response level calculated
- Real computedRelLevel
  output reliability level calculated for RIA and 1st-order PMA
- Real computedGenRelLevel
  output generalized reliability level calculated for 2nd-order PMA
- RealVector fnGradX
  actual x-space gradient for current function from most recent response evaluation
- RealVector fnGradU
  u-space gradient for current function updated from fnGradX and Jacobian dx/du
- RealSymMatrix fnHessX
  actual x-space Hessian for current function from most recent response evaluation
- RealSymMatrix fnHessU
  u-space Hessian for current function updated from fnHessX and Jacobian dx/du
- RealVector kappaU
  principal curvatures derived from eigenvalues of orthonormal transformation of fnHessU
- RealVector fnValsMeanX
  response function values evaluated at mean x
- RealMatrix fnGradsMeanX
  response function gradients evaluated at mean x
- RealSymMatrixArray fnHessiansMeanX
  response function Hessians evaluated at mean x
- RealVector ranVarMeansX
  vector of means for all uncertain random variables in x-space
- RealVector ranVarStdDevsX
  vector of std deviations for all uncertain random variables in x-space
- RealVector ranVarMeansU
  vector of means for all uncertain random variables in u-space
- bool initialPtUserSpec
  flag indicating user specification of (any portion of) initialPtU
- RealVector initialPtUSpec
  user specification or default initial guess for local optimization
- RealVector initialPtU
  current starting point for MPP searches in u-space
- RealVector mostProbPointX
  location of MPP in x-space
- RealVector mostProbPointU
  location of MPP in u-space
- RealVectorArray prevMPPULev0
  array of converged MPP’s in u-space for level 0. Used for warm-starting initialPtU within RBDO.
- RealMatrix prevFnGradDLev0
matrix of limit state sensitivities w.r.t. inactive/design variables for level 0. Used for warm-starting initialPtU within RBDO.

- **RealMatrix** `prevFnGradULev0`
  matrix of limit state sensitivities w.r.t. active/uncertain variables for level 0. Used for warm-starting initialPtU within RBDO.

- **RealVector** `prevICVars`
  previous design vector. Used for warm-starting initialPtU within RBDO.

- **ShortArray** `prevCumASVLev0`
  accumulation (using \(\|=\)) of all previous design ASV’s from requested finalStatistics. Used to detect availability of `prevFnGradDLev0` data for warm-starting initialPtU within RBDO.

- **bool** `npsolFlag`
  flag representing the optimization MPP search algorithm selection (NPSOL SQP or OPT++ NIP)

- **bool** `warmStartFlag`
  flag indicating the use of warm starts

- **bool** `nipModeOverrideFlag`
  flag indicating the use of move overrides within OPT++ NIP

- **bool** `curvatureDataAvailable`
  flag indicating that sufficient data (i.e., fnGradU, fnHessU, mostProbPointU) is available for computing principal curvatures

- **bool** `kappaUpdated`
  track when kappaU requires updating via principal_curvatures()

- **short** `integrationOrder`
  integration order (1 or 2) provided by integration specification

- **short** `secondOrderIntType`
  type of second-order integration: Breitung, Hohenbichler-Rackwitz, or Hong

- **Real** `curvatureThresh`
  cut-off value for \(1/\sqrt{}\) term in second-order probability corrections.

- **short** `taylorOrder`
  order of Taylor series approximations (1 or 2) in MV/AMV/AMV+ derived from hessian type

- **RealMatrix** `impFactor`
  importance factors predicted by MV

- **int** `npsolDerivLevel`
  derivative level for NPSOL executions (1 = analytic grads of objective fn, 2 = analytic grads of constraints, 3 = analytic grads of both).

- **unsigned short** `warningBits`
  set of warnings accumulated during execution

### Static Private Attributes

- **static** `NonDLocalReliability * nondLocRelInstance`
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
Additional Inherited Members

13.118.1 Detailed Description

Class for the reliability methods within DAKOTA/UQ.

The NonDLocalReliability class implements the following reliability methods through the support of different limit state approximation and integration options: mean value (MVFOSm/MVSOsM), advanced mean value method (AMV, AMV\(\wedge 2\)) in x- or u-space, iterated advanced mean value method (AMV+, AMV\(\wedge 2+\)) in x- or u-space, two-point adaptive nonlinearity approximation (TANA) in x- or u-space, first order reliability method (FORM), and second order reliability method (SORM). All options except mean value employ an optimizer (currently NPSOL SQP or OPT++ NIP) to solve an equality-constrained optimization problem for the most probable point (MPP). The MPP search may be formulated as the reliability index approach (RIA) for mapping response levels to reliabilities/probabilities or as the performance measure approach (PMA) for performing the inverse mapping of reliability/probability levels to response levels.

13.118.2 Member Function Documentation

```
void RIA_objective_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]
```

static function used as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of (norm u)\(\wedge 2\).

This function recasts a G(u) response set (already transformed and approximated in other recursions) into an RIA objective function.

References Response::active_set_request_vector(), Variables::continuous_variables(), Response::function_gradient_view(), Response::function_hessian_view(), and Response::function_value().

Referenced by NonDLocalReliability::mpp_search().

```
void RIA_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]
```

static function used as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of G(u) = response level.

This function recasts a G(u) response set (already transformed and approximated in other recursions) into an RIA equality constraint.

References Response::active_set_request_vector(), Response::function_gradient(), Response::function_gradient_view(), Response::function_hessian(), Response::function_value(), NonDLocalReliability::nondLocRelInstance, NonDReliability::requestedTargetLevel, and NonDReliability::respFnCount.

Referenced by NonDLocalReliability::mpp_search().

```
void PMA_objective_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]
```

static function used as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of G(u).

This function recasts a G(u) response set (already transformed and approximated in other recursions) into an PMA objective function.
void PMA_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]

static function used as the constraint function in the first-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of $(\text{norm } u)^2 = (\beta_{bar})^2$.

This function recasts a $G(u)$ response set (already transformed and approximated in other recursions) into a first-order PMA equality constraint on reliability index beta.

References Dakota::abort_handler(), Response::active_set_request_vector(), NonD::cdfFlag, NonDLocalReliability::computedGenRelLevel, NonDLocalReliability::computedRelLevel, Variables::continuous_variables(), NonDLocalReliability::dp2_dbeta_factor(), NonDLocalReliability::fnGradU, NonDLocalReliability::fnHessU, Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), NonDLocalReliability::nondLocRelInstance, NonDReliability::mppSearchType, NonDReliability::pmaMaximizeG, NonDReliability::respFnCount, and NonDLocalReliability::update_pma_maximize.

Referenced by NonDLocalReliability::mpp_search().

void PMA2_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]

static function used as the constraint function in the second-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of $\beta^* = \beta_{star} - \beta_{bar}$.

This function recasts a $G(u)$ response set (already transformed and approximated in other recursions) into a second-order PMA equality constraint on generalized reliability index beta-star.

References Dakota::abort_handler(), Response::active_set_request_vector(), NonD::cdfFlag, NonDLocalReliability::computedGenRelLevel, NonDLocalReliability::computedRelLevel, Variables::continuous_variables(), NonDLocalReliability::dp2_dbeta_factor(), NonDLocalReliability::fnGradU, NonDLocalReliability::fnHessU, Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), NonDLocalReliability::nondLocRelInstance, NonDReliability::mppSearchType, NonDReliability::nondLocRelInstance, NonDLocalReliability::probability(), NonDLocalReliability::probability(), NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, and NonDLocalReliability::signed_norm.

Referenced by NonDLocalReliability::mpp_search().

void initial_taylor_series ( ) [private]

convenience function for performing the initial limit state Taylor-series approximation.

An initial first- or second-order Taylor-series approximation is required for MV/AMV/AMV+/TANA or for the case where momentStats (from MV) are required within finalStatistics for subIterator usage of NonDLocalReliability.

References Response::active_set_request_vector(), Iterator::activeSet, Model::component_parallel_mode(), Model::continuous_variables(), Model::current_response(), Model::evaluate(), NonD::finalMomentsType, NonD::finalStatistics, NonDLocalReliability::fnGradsMeanX, NonDLocalReliability::fnHessiansMeanX, NonDLocalReliability::fnValsMeanX, Response::function_gradients(), Response::function_hessians(), Response::function_values(), Model::hessian_type(), Iterator::iteratedModel, NonD::momentStats, NonDReliability::mppSearchType, NonD::natafTransform, Analyzer::numFunctions, NonD::numUncertainVars, NonDLocalReliability::ranVarMeansX, NonD-
LocalReliability::ranVarStdDevsX, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, Iterator::subIteratorFlag, NonDLocalReliability::taylorOrder, and NonDReliability::uSpaceModel.

Referenced by NonDLocalReliability::mean_value(), and NonDLocalReliability::mpp_search().

```cpp
void initialize_class_data() [private]
```

convenience function for initializing class scope arrays

Initialize class-scope arrays and perform other start-up activities, such as evaluating median limit state responses.

Referenced Response::active_set_derivative_vector(), NonD::finalStatistics, NonDReliability::mppModel, NonD::natafTransform, Analyzer::numFunctions, NonDReliability::numRelAnalyses, NonD::numUncertainVars, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevFnGradDLev0, NonDLocalReliability::prevFnGradULev0, NonDLocalReliability::prevMPPULev0, NonDLocalReliability::ranVarMeansX, Iterator::subIteratorFlag, Model::update_from_subordinate_model(), and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void initialize_level_data() [private]
```

convenience function for initializing/warm starting MPP search data for each response function prior to level 0

For a particular response function prior to the first z/p/beta level, initialize/warm-start optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/-U, and fnHessX/U).

Referenced Iterator::activeSet, NonDLocalReliability::assign_mean_data(), Model::component_parallel_mode(), NonDLocalReliability::computedRespLevel, Model::continuous_variable_ids(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), NonDLocalReliability::curvatureDataAvailable, Model::evaluate(), NonDLocalReliability::fnGradU, NonDLocalReliability::fnGradX, NonDLocalReliability::fnHessU, NonDLocalReliability::fnHessX, Response::function_gradient_copy(), Response::function_hessian(), Response::function_value(), Model::inactive_continuous_variables(), NonDLocalReliability::initialPtU, NonDLocalReliability::initialPtUSpec, Iterator::iteratedModel, NonDLocalReliability::kappaUpdated, NonDLocalReliability::mostProbPointU, NonDLocalReliability::mostProbPointX, NonDReliability::mppSearchType, NonD::natafTransform, NonDReliability::numRelAnalyses, NonD::numUncertainVars, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevCov, NonDLocalReliability::prevGradDLev0, NonDLocalReliability::prevGradULev0, NonDLocalReliability::prevICVars, NonDLocalReliability::prevMPULev0, ActiveSet::request_value(), ActiveSet::request_values(), NonD::requestedRespLevels, NonDReliability::respFnCount, Iterator::subIteratorFlag, Model::surrogate_function_indices(), NonDLocalReliability::taylorOrder, NonDLocalReliability::update_limit_state_surrogate(), NonDReliability::uSpaceModel, and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void initialize_mpp_search_data() [private]
```

convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function

For a particular response function at a particular z/p/beta level, warm-start or reset the optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/-U, and fnHessX/U).

References NonDLocalReliability::assign_mean_data(), NonD::computedGenRelLevels, NonD::computedRelLevels, NonDLocalReliability::fnGradU, Model::hessian_type(), NonDLocalReliability::initialPtU, NonDLocalReliability::initialPtUSpec, NonDLocalReliability::integrationOrder, Iterator::iteratedModel, NonDReliability::level-
Count, NonDLocalReliability::mostProbPointU, NonD::numUncertainVars, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, NonDLocalReliability::taylorOrder, and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

```
void update_mpp_search_data ( const Variables & vars_star, const Response & resp_star ) [private]
```

convenience function for updating MPP search data for each z/p/beta level for each response function

Includes case-specific logic for updating MPP search data for the AMV/AMV+/TANA/NO APPROX methods.

Referenced by NonDLocalReliability::mpp_search().

```
void update_level_data ( ) [private]
```

convenience function for updating z/p/beta level data and final statistics following MPP convergence

Updates computedRespLevels/computedProbLevels/computedRelLevels, finalStatistics, warm start, and graphics data.

Referenced by NonDLocalReliability::mpp_search().

void dg_ds_eval ( const RealVector & x_vars, const RealVector & fn_grad_x, RealVector & final_stat_grad ) [private]

convenience function for evaluating dg/ds

Computes dg/ds where s = design variables. Supports potentially overlapping cases of design variable augmentation and insertion.

References Response::active_set_derivative_vector(), Iterator::activeSet, Model::all_continuous_variable_ids(), Model::component_parallel_mode(), Dakota::contains(), Model::continuous_variable_ids(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), ActiveSet::derivative_vector(), Model::evaluate(), Non-D::finalStatistics, Response::function_gradient_copy(), Response::function_gradients(), Model::inactive_continuous_variable_ids(), Iterator::iteratedModel, NonDReliability::mppSearchType, Iterator::primaryACVarMapIndices, ActiveSet::request_value(), ActiveSet::request_values(), NonDReliability::respFnCount, Iterator::secondaryACVarMapTargets, and NonDReliability::uSpaceModel.

Referenced by NonDLocalReliability::mean_value(), NonDLocalReliability::mpp_search(), and NonDLocalReliability::update_level_data().

Real dp2_dbeta_factor ( Real beta, bool cdf_flag ) [private]

compute factor for derivative of second-order probability with respect to reliability index (from differentiating BREITUNG or HOHENRACK expressions)

Compute sensitivity of second-order probability w.r.t. beta for use in derivatives of p_2 or beta* w.r.t. auxilliary parameters s (design, epistemic) or derivatives of beta* w.r.t. u in PMA2_constraint_eval().

References Dakota::abort_handler(), NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::curvatureThresh, NonDLocalReliability::kappaU, NonD::numUncertainVars, NonDLocalReliability::probability(), NonDLocalReliability::scale_curvature(), NonDLocalReliability::secondOrderIntType, and NonDLocalReliability::warningBits.

Referenced by NonDLocalReliability::PMA2_constraint_eval(), and NonDLocalReliability::update_level_data().

Real probability ( Real beta, bool cdf_flag, const RealVector & mpp_u, const RealVector & fn_grad_u, const RealSymMatrix & fn_hess_u ) [private]

Convert provided reliability to probability using either a first-order or second-order integration.

Converts beta into a probability using either first-order (FORM) or second-order (SORM) integration. The SORM calculation first calculates the principal curvatures at the MPP (using the approach in Ch. 8 of Haldar & Mahadevan), and then applies correction formulations from the literature (Breitung, Hohenbichler-Rackwitz, or Hong).

References NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::curvatureThresh, NonDAdaptImpSampling::final_probability(), NonDReliability::importanceSampler, NonDLocalReliability::integrationOrder, NonDReliability::integrationRefinement, Iterator::iterator_rep(), NonDLocalReliability::kappaU, NonDLocalReliability::kappaUpdated, NonDLocalReliability::levelCount, Iterator::methodPCIter, NonD::miPLIndex, NonD::numUncertainVars, Iterator::outputLevel, NonDLocalReliability::principal_curvatures(), NonDLocalReliability::probability(), NonD::requestedRespLevels, NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, Iterator::run(), NonDLocalReliability::scale_curvature(), NonDLocalReliability::secondOrderIntType, NonDLocalReliability::warningBits, and Dakota::write_precision.

The documentation for this class was generated from the following files:

- NonDLocalReliability.hpp
- NonDLocalReliability.cpp
13.119  NonDLocalSingleInterval Class Reference

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDLocalSingleInterval:

```
NonDLocalSingleInterval
    NonDLocalInterval
        NonDInterval
            NonD
                Analyzer
                    Iterator
```

Public Member Functions

- **NonDLocalSingleInterval (ProblemDescDB &problem_db, Model &model)**  
  constructor
- **~NonDLocalSingleInterval ()**  
  destructor

Protected Member Functions

- **void initialize ()**  
  perform any required initialization
- **void post_process_cell_results (bool maximize)**  
  post-process a cell minimization/maximization result

Private Attributes

- **size_t statCntr**  
  counter for finalStatistics

Additional Inherited Members

13.119.1  Detailed Description

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The **NonDLocalSingleInterval** class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval
13.20 NonDMultilevelPolynomialChaos Class Reference

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

Inheritance diagram for NonDMultilevelPolynomialChaos:

```
   Iterator
    |   
   Analyzer
    |   
   NonD
    |   
   NonDExpansion
    |   
   NonDPolynomialChaos
    |   
NonDMultilevelPolynomialChaos
```

**Public Member Functions**

- `NonDMultilevelPolynomialChaos (ProblemDescDB &problem_db, Model &model)`
  *standard constructor*

- `NonDMultilevelPolynomialChaos (Model &model, short exp_coeffs_approach, const UShortArray &num_int_seq, const RealVector &dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs)`
  *alternate constructor for numerical integration (tensor, sparse, cubature)*

- `NonDMultilevelPolynomialChaos (unsigned short method_name, Model &model, short exp_coeffs_approach, const UShortArray &exp_order_seq, const RealVector &dim_pref, const SizetArray &colloc_pts_seq, Real colloc_ratio, const SizetArray &pilot, int seed, short u_space_type, bool piecewise_basis, bool use_derivs, bool cv_flag, const String &import_build_pts_file, unsigned short import_build_format, bool import_build_active_only)`
  *alternate constructor for regression (least squares, CS, OLI)*

- `~NonDMultilevelPolynomialChaos ()`
  *destructor*

**Protected Member Functions**

- `void core_run ()`
  *perform a forward uncertainty propagation using PCE/SC methods*
void increment_specification_sequence ()
    increment the input specification sequence (PCE only)

void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
    print the final statistics

void multilevel_regression ()
    option within core_run() for multilevel allocation of samples (mirroring NonDMultilevelSampling for PCE regression)

void increment_sample_sequence (size_t new_samp, size_t total_samp, size_t lev)
    increment the sequence in numSamplesOnModel for multilevel_regression()

Private Attributes

- UShortArray expOrderSeqSpec
  user specification for expansion order (array for multifidelity)
- SizetArray collocPtsSeqSpec
  user specification for collocation points (array for multifidelity)
- SizetArray expSamplesSeqSpec
  user specification for expansion samples (array for multifidelity)
- UShortArray quadOrderSeqSpec
  user request of quadrature order
- UShortArray ssgLevelSeqSpec
  user request of sparse grid level
- size_t sequenceIndex
  sequence index for {expOrder,collocPts,expSamples}SeqSpec
- Real kappaEstimatorRate
  rate parameter for estimator variance in ML PCE
- Real gammaEstimatorScale
  scale parameter for estimator variance in ML PCE
- SizetArray NLev
  number of samples allocated to each level of a discretization hierarchy within multilevel regression
- SizetArray pilotSamples
  number of initial samples specified by the user
- Real equivHFEvals
  equivalent number of high fidelity evaluations accumulated using samples across multiple model forms and/or discretization levels

Additional Inherited Members

13.120.1 Detailed Description

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

The NonDMultilevelPolynomialChaos class uses a polynomial chaos expansion (PCE) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the OrthogPolyApproximation class to manage multiple types of orthogonal polynomials within a Wiener-Askey scheme to PCE. It supports PCE coefficient estimation via sampling, quadrature, point-collocation, and file import.
13.120.2 Constructor & Destructor Documentation

NonDMultilevelPolynomialChaos ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Dakota::abort_handler(), Response::active_set(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDMultilevelPolynomialChaos::collocPtsSeqSpec, ParallelLibrary::command_line_check(), NonDPolynomialChaos::config_expansion_orders(), NonDPolynomialChaos::config_expectation(), NonDPolynomialChaos::config_integration(), NonDPolynomialChaos::config_regression(), NonDExpansion::construct_expansion_sampler(), NonDPolynomialChaos::cubIntSpec, Model::current_response(), NonDPolynomialChaos::dimPrefSpec, NonDMultilevelPolynomialChaos::expOrderSeqSpec, NonDMultilevelPolynomialChaos::expSamplesSeqSpec, ProblemDescDB::get_real(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_usa(), ProblemDescDB::get_ushort(), NonDPolynomialChaos::importApproxActiveOnly, NonDPolynomialChaos::importApproxPointsFile, NonDPolynomialChaos::importBuildActiveOnly, NonDPolynomialChaos::importBuildFormat, NonDPolynomialChaos::importBuildPointsFile, NonDExpansion::initialize_random(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, Iterator::probDescDB, NonDMultilevelPolynomialChaos::quadOrderSeqSpec, ActiveSet::request_values(), NonDPolynomialChaos::resolve_inputs(), NonDMultilevelPolynomialChaos::sequenceIndex, NonD::transform_model(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDMultilevelPolynomialChaos ( Model & model, short exp_coeffs_approach, const UShortArray & exp_order_seq, const RealVector & dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs )

alternate constructor for numerical integration (tensor, sparse, cubature)

This constructor is used for helper iterator instantiation on the fly that employ numerical integration (quadrature, sparse grid, cubature).

References Dakota::abort_handler(), Response::active_set(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDMultilevelPolynomialChaos::config_integration(), NonDPolynomialChaos::cubIntSpec, Model::current_response(), NonDExpansion::initialize_random(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, NonDMultilevelPolynomialChaos::quadOrderSeqSpec, ActiveSet::request_values(), NonDPolynomialChaos::resolve_inputs(), NonDMultilevelPolynomialChaos::sequenceIndex, NonD::transform_model(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDMultilevelPolynomialChaos ( unsigned short method_name, Model & model, short exp_coeffs_approach, const UShortArray & exp_order_seq, const RealVector & dim_pref, const SizetArray & colloc_pts_seq, Real colloc_ratio, const SizetArray & pilot, int seed, short u_space_type, bool piecewise_basis, bool use_derivs, bool cv_flag, const String & import_build_pts_file, unsigned short import_build_format, bool import_build_active_only )

alternate constructor for regression (least squares, CS, OLI)

This constructor is used for helper iterator instantiation on the fly that employ regression (least squares, CS, OLI).

References Response::active_set(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDMultilevelPolynomialChaos::collocPtsSeqSpec, NonDPolynomialChaos::config_expansion_orders(), NonDPolynomialChaos::config_integration(), Model::current_response(), NonDPolynomialChaos::dimPrefSpec, NonDMultilevelPolynomialChaos::expOrderSeqSpec, NonDExpansion::initialize_random(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, ActiveSet::request_values(), NonD::transform_model(), NonDExpansion::uSpaceModel, and NonD::uSpaceType.
Chaos::resolve_inputs(), NonDMultilevelPolynomialChaos::sequenceIndex, NonD::transform_model(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

The documentation for this class was generated from the following files:

- NonDMultilevelPolynomialChaos.hpp
- NonDMultilevelPolynomialChaos.cpp

### 13.121 NonDMultilevelSampling Class Reference

Performs Multilevel Monte Carlo sampling for uncertainty quantification.

Inheritance diagram for NonDMultilevelSampling:

```
Analyzer
NonD
NonDSampling
NonDMultilevelSampling
```

#### Public Member Functions

- **NonDMultilevelSampling** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- **~NonDMultilevelSampling** ()
  
  *destructor*

- **bool resize** ()
  
  *reinitializes iterator based on new variable size*

#### Protected Member Functions

- **void pre_run** ()
  
  *pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- **void core_run** ()

- **void post_run** (std::ostream &s)
  
  *post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way*

- **void print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  
  *print the final iterator results*
Private Member Functions

- void `multilevel_mc_Ysum` (size_t model_form)
  - Perform multilevel Monte Carlo across the discretization levels for a particular model form using discrepancy accumulators (\(\text{sum}_Y\))

- void `multilevel_mc_Qsum` (size_t model_form)
  - Perform multilevel Monte Carlo across the discretization levels for a particular model form using QoI accumulators (\(\text{sum}_Q\))

- void `control_variate_mc` (const SizetSizetPair &lf_form_level, const SizetSizetPair &hf_form_level)
  - Perform control variate Monte Carlo across two model forms.

- void `multilevel_control_variate_mc_Ycorr` (size_t lf_model_form, size_t hf_model_form)
  - Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for \(Y\) (LH correlations for level discrepancies)

- void `multilevel_control_variate_mc_Qcorr` (size_t lf_model_form, size_t hf_model_form)
  - Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for \(Q\) (LH correlations for QoI)

- void `shared_increment` (size_t iter, size_t lev)
  - Perform a shared increment of LF and HF samples for purposes of computing/updating the evaluation ratio and the MSE ratio

- bool `lf_increment` (Real avg_eval_ratio, const SizetArray &N_lf, const SizetArray &N_hf, size_t iter, size_t lev)
  - Perform final LF sample increment as indicated by the evaluation ratio

- void `initialize_ml_Ysums` (IntRealMatrixMap &sum_Y, size_t num_lev)
  - Initialize the ML accumulators for computing means, variances, and covariances across fidelity levels

- void `initialize_ml_Qsums` (IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, IntIntPairRealMatrixMap &sum_QlQlm1, size_t num_lev)
  - Initialize the ML accumulators for computing means, variances, and covariances across fidelity levels

  - Initialize the CV accumulators for computing means, variances, and covariances across fidelity levels

- void `initialize_mlcv_sums` (IntRealMatrixMap &sum_Ll, IntRealMatrixMap &sum_Llm1, IntRealMatrixMap &sum_Llrefined, IntRealMatrixMap &sum_Llm1refined, IntRealMatrixMap &sum_Hl, IntRealMatrixMap &sum_Hlm1, IntRealMatrixMap &sum_LlLlm1, IntRealMatrixMap &sum_LlLlm1refined, IntRealMatrixMap &sum_HlHlm1, IntRealMatrixMap &sum_HlHlm1Llm1, IntRealMatrixMap &sum_HlHlm1Llm1refined, IntRealMatrixMap &sum_HlHlm1Hlm1, IntRealMatrixMap &sum_HlHlm1Hlm1refined, size_t num_ml_lev, size_t num_cv_lev)
  - Initialize the MLCV accumulators for computing means, variances, and covariances across fidelity levels

- void `accumulate_offsets` (RealVector &mu)
  - Accumulate initial approximation to mean vector for use as offsets in subsequent accumulations

- void `accumulate_ml_Qsums` (IntRealMatrixMap &sum_Q, size_t lev, const RealVector &offset, SizetArray &num_Q)
update running QoI sums for one model (sum_Q) using set of model evaluations within allResponses; used for level 0 from other accumulators

- void accumulate_ml_Ysums (IntRealMatrixMap &sum_Y, RealMatrix &sum_YY, size_t lev, const RealVector &offset, SizetArray &num_Y)

  update accumulators for multilevel telescoping running sums using set of model evaluations within allResponses

- void accumulate_ml_Qsums (IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, IntIntPairRealMatrixMap &sum_QlQlm1, size_t lev, const RealVector &offset, SizetArray &num_Q)

  update running QoI sums for two models (sum_Ql, sum_Qlm1) using set of model evaluations within allResponses

- void accumulate_cv_Ysums (IntRealVectorMap &sum_L, const RealVector &offset, SizetArray &num_L)

  update running sums for one model (sum_L) using set of model evaluations within allResponses


  update running sums for two models (sum_L, sum_H, and sum_LH) from set of low/high fidelity model evaluations within allResponses

- void accumulate_mlcv_Qsums (IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, size_t lev, const RealVector &offset, SizetArray &num_Q)

  update running QoI sums for one model at two levels (sum_Ql, sum_Qlm1) using set of model evaluations within allResponses

- void accumulate_mlcv_Ysums (IntRealMatrixMap &sum_Y, size_t lev, const RealVector &offset, SizetArray &num_Y)

  update running discrepancy sums for one model (sum_Y) using set of model evaluations within allResponses


  update running QoI sums for two models (sum_L, sum_H, sum_LL, sum_LH, and sum_HH) from set of low/high fidelity model evaluations within {lf,hf}resp_map; used for level 0 from other accumulators


  update running two-level discrepancy sums for two models (sum_L, sum_H, sum_LL, sum_LH, and sum_HH) from set of low/high fidelity model evaluations within {lf,hf}resp_map

- void accumulate_mlcv_Qsums (const IntResponseMap &lf_resp_map, const IntResponseMap &hf_resp_map, IntRealMatrixMap &sum_Ll, IntRealMatrixMap &sum_Llm1, IntRealMatrixMap &sum_Llm1_reined, IntRealMatrixMap &sum_Llm1_Hlm1, IntRealMatrixMap &sum_Hl, IntRealMatrixMap &sum_HlLlm1, IntRealMatrixMap &sum_HlLlm1_Llm1, IntRealMatrixMap &sum_HlLlm1_Hlm1, IntRealMatrixMap &sum_HlLlm1_Hlm1, size_t lev, const RealVector &lf_offset, const RealVector &hf_offset, SizetArray &num_L, SizetArray &num_H)

  update running QoI sums for two models and two levels from set of low/high fidelity model evaluations within {lf,hf}resp_map

- Real eval_ratio (const RealVector &sum_L_shared, const RealVector &sum_H, const RealVector &sum_LL, const RealVector &sum_LH, const RealVector &sum_HH, Real cost_ratio, const SizetArray &N_shared, RealVector &var_H, RealVector &rho2_LH)
compute the LF/HF evaluation ratio, averaged over the QoI

- Real eval\_ratio (RealMatrix &sum\_L\_shared, RealMatrix &sum\_H, RealMatrix &sum\_LL, RealMatrix &sum\_LH, RealMatrix &sum\_HH, Real cost\_ratio, size\_t lev, const SizetArray &N\_shared, RealMatrix &var\_H, RealMatrix &rho2\_LH)

compute the LF/HF evaluation ratio, averaged over the QoI

- Real eval\_ratio (RealMatrix &sum\_Ll, RealMatrix &sum\_Llm1, RealMatrix &sum\_Hl, RealMatrix &sum\_Hlm1, RealMatrix &sum\_Ll\_Ll, RealMatrix &sum\_Ll\_Llm1, RealMatrix &sum\_Llm1\_Llm1, RealMatrix &sum\_Hl\_Ll, RealMatrix &sum\_Hl\_Llm1, RealMatrix &sum\_Hlm1\_Ll, RealMatrix &sum\_Hlm1\_Llm1, RealMatrix &sum\_HI\_HI, RealMatrix &sum\_HI\_Hlm1, RealMatrix &sum\_HI\_Hlm1\_Ll, RealMatrix &sum\_HI\_Hlm1\_Llm1, RealMatrix &sum\_HI\_Hlm1\_Llm1, RealMatrix &sum\_HI\_Hlm1\_Llm1, RealMatrix &sum\_HI\_Hlm1, Real cost\_ratio, size\_t lev, const SizetArray &N\_shared, RealMatrix &var\_YH, RealMatrix &rho\_dot2\_LH)

compute the LF/HF evaluation ratio, averaged over the QoI

- Real MSE\_ratio (Real avg\_eval\_ratio, const RealVector &var\_H, const RealVector &rho2\_LH, size\_t iter, const SizetArray &N\_hf)

compute ratio of MC and CVMC mean squared errors, averaged over the QoI


compute control variate parameters for CVMC and estimate raw moments


apply control variate parameters for MLCVMC to estimate raw moment contributions


apply control variate parameters for MLCVMC to estimate raw moment contributions

- void compute\_control (Real sum\_L, Real sum\_H, Real sum\_LL, Real sum\_LH, size\_t N\_shared, Real &beta)

compute scalar control variate parameters

- void compute\_control (Real sum\_L, Real sum\_H, Real sum\_LL, Real sum\_LH, Real sum\_HH, size\_t N\_shared, Real &var\_H, Real &rho2\_LH)

compute scalar variance and correlation parameters for control variates

- void compute\_control (Real sum\_Ll, Real sum\_Llm1, Real sum\_Hl, Real sum\_Hlm1, Real sum\_Ll\_Llm1, Real sum\_Hl\_Llm1, Real sum\_Hlm1\_Llm1, Real sum\_HI\_HI, Real sum\_HI\_Hlm1, Real sum\_HI\_Hlm1\_Ll, Real sum\_HI\_Hlm1\_Llm1, size\_t N\_shared, Real &var\_YH, Real &rho\_dot2\_LH, Real &beta\_dot, Real &gamma)

compute scalar control variate parameters

- void compute\_control (const RealVector &sum\_L, const RealVector &sum\_H, const RealVector &sum\_LL, const RealVector &sum\_LH, const SizetArray &N\_shared, RealVector &beta)

compute vector control variate parameters

- void compute\_control (const RealVector &sum\_L, const RealVector &sum\_H, const RealVector &sum\_LL, const RealVector &sum\_LH, const RealVector &sum\_HH, const SizetArray &N\_shared, RealVector &var\_H, RealVector &rho2\_LH)
compute vector variance and correlation parameters for control variates

- void compute_control (const RealMatrix &sum_L, const RealMatrix &sum_H, const RealMatrix &sum_LL, const RealMatrix &sum_LH, const SizetArray &N_shared, size_t lev, RealVector &beta)

compute matrix control variate parameters

- void compute_control (const RealMatrix &sum_Ll, const RealMatrix &sum_Llm1, const RealMatrix &sum_Hl, const RealMatrix &sum_Hlm1, const RealMatrix &sum_LlLlm1, const RealMatrix &sum_HlLlm1, const RealMatrix &sum_HlLl, const RealMatrix &sum_HlHlm1, const RealMatrix &sum_HI_Hlm1, const RealMatrix &sum_HI_Hl, const SizetArray &N_shared, size_t lev, RealVector &beta, RealVector &beta_dot, RealVector &gamma)

compute matrix control variate parameters

- void apply_control (const RealVector &sum_H, Real sum_L_shared, size_t N_shared, Real sum_L_refined, size_t N_refined, RealVector &beta, Real &H_raw_mom)
- void apply_control (const RealVector &sum_HI, const RealVector &sum_HI_Hlm1, const RealVector &sum_Ll, const RealVector &sum_Llm1, const SizetArray &N_shared, const RealVector &sum_Ll_refined, const RealVector &sum_Llm1_refined, const SizetArray &N_refined, const RealVector &beta, RealVector &H_raw_mom)
- void apply_control (const RealMatrix &sum_Hl, const RealMatrix &sum_HI_Hlm1, const RealMatrix &sum_HI_Hl, const RealMatrix &sum_HI_Hl_Llm1, const RealMatrix &sum_HI_Hl_Llm1, const RealMatrix &sum_HI_Hl_Hlm1, const SizetArray &N_shared, size_t lev, const RealVector &beta, const RealVector &gamma, RealVector &H_raw_mom)

apply control variate parameter (beta) to approximate HF moment

- void export_all_samples (String root_prepend, const Model &model, size_t iter, size_t lev)

export allSamples to tagged tabular file

- void convert_moments (const RealMatrix &raw_mom, RealMatrix &final_mom)

convert uncentered raw moments (multilevel expectations) to standardized moments

- void compute_error_estimates (IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, IntIntPairRealMatrixMap &sum_QlQl, IntIntPairRealMatrixMap &sum_QlQlm1, Sizet2DArray &num_Q)

populate finalStatErrors for MLMC based on Q sums

- Real aggregate_variance_Ysum (const RealVector *sum_Y, const RealVector *sum_YY, const SizetArray &N_lq)
- Real aggregate_variance_Qsum (const RealVector *sum_Ql, const RealVector *sum_Qlm1, const RealVector *sum_QlQl, const RealVector *sum_QlQlm1, const RealVector *sum_Qlm1Qlm1, const SizetArray &N_lq)

compute variance from sum accumulators

- Real aggregate_variance_Ysum (const Real *sum_Y, const Real *sum_YY, const SizetArray &N_lq)
- Real aggregate_variance_Qsum (const Real *sum_Ql, const Real *sum_Qlm1, const Real *sum_QlQl, const Real *sum_QlQlm1, const Real *sum_Qlm1Qlm1, const SizetArray &N_lq, size_t lev)
sum up variances across QoI (using sum_YY with means from sum_Y)

- Real aggregate_mse_Yvar (const Real *var_Y, const SizetArray &N_l)
  sum up Monte Carlo estimates for mean squared error (MSE) across QoI using discrepancy variances

- Real aggregate_mse_Ysum (const Real *sum_Y, const Real *sum_YY, const SizetArray &N_l)
  sum up Monte Carlo estimates for mean squared error (MSE) across QoI using discrepancy sums

- Real aggregate_mse_Qsum (const Real *sum_Ql, const Real *sum_Qlm1, const Real *sum_QlQl, const Real *sum_QlQlm1, const SizetArray &N_l, size_t lev)
  sum up Monte Carlo estimates for mean squared error (MSE) across QoI using discrepancy sums

- void uncentered_to_centered (Real rm1, Real rm2, Real rm3, Real rm4, Real &cm1, Real &cm2, Real &cm3, Real &cm4, size_t Nlq) const
  convert uncentered (raw) moments to centered moments; biased estimators

- void uncentered_to_centered (Real rm1, Real rm2, Real rm3, Real rm4, Real &cm1, Real &cm2, Real &cm3, Real &cm4) const
  convert uncentered (raw) moments to centered moments; unbiased estimators

- void centered_to_standard (Real cm1, Real cm2, Real cm3, Real cm4, Real &sm1, Real &sm2, Real &sm3, Real &sm4) const
  convert centered moments to standardized moments

- void check_negative (Real &cm) const
  detect, warn, and repair a negative central moment (for even orders)

- Real sum (const Real *vec, size_t vec_len) const
  compute sum of a set of observations

- Real average (const Real *vec, size_t vec_len) const
  compute average of a set of observations

- Real average (const RealVector &vec) const
  compute average of a set of observations

- Real average (const SizetArray &sa) const
  compute average of a set of observations

- size_t one_sided_delta (Real current, Real target)
  compute a one-sided sample increment to move current sampling level to a new target

Private Attributes

- Sizet3DArray NLev
  total number of successful sample evaluations (excluding faults) for each model form, discretization level, and QoI

- SizetArray pilotSamples
  store the pilot_samples input specification, prior to run-time invocation of load_pilot_sample()

- RealVector mcMSEIter0
  mean squared error of mean estimator from pilot sample MC on HF model

- Real equivHFEvals
  equivalent number of high fidelity evaluations accumulated using samples across multiple model forms and/or discretization levels

- bool finalCVRefinement
  if defined, complete the final CV refinement when terminating MLCV based on maxIterations (the total number of refinements beyond the pilot sample will be one more for CV than for ML). This approach is consistent with normal termination based on \( \| \text{norm}(\text{delta}_N_{hf}) \| = 0 \).
13.121. NONDMULTILEVELSAMPLING CLASS REFERENCE

- bool exportSampleSets
  
  if defined, export each of the sample increments in ML, CV, MLCV using tagged tabular files
- unsigned short exportSamplesFormat

  format for exporting sample increments using tagged tabular files

Additional Inherited Members

13.121.1 Detailed Description

Performs Multilevel Monte Carlo sampling for uncertainty quantification.

Multilevel Monte Carlo (MLMC) is a variance-reduction technique that utilizes lower fidelity simulations that have response QoI that are correlated with the high-fidelity response QoI.

13.121.2 Constructor & Destructor Documentation

NonDMultilevelSampling ( ProblemDescDB & problem db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, NonDMultilevelSampling::NLev, NonDMultilevelSampling::pilotSamples, NonDSampling::sampleType, Model::subordinate_models(), Model::surrogate_response_mode(), and Model::surrogate_type().

13.121.3 Member Function Documentation

void pre_run ( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from NonD.

References NonDMultilevelSampling::NLev, Analyzer::numFunctions, and NonDSampling::pre_run().

void core_run ( ) [protected], [virtual]

The primary run function manages the general case: a hierarchy of model forms (from the ordered model fidelities within a HierarchSurrModel), each of which may contain multiple discretization levels.

Reimplemented from Iterator.

References Dakota::NPOS, NonDMultilevelSampling::control_variate_mc(), NonDMultilevelSampling::multilevel_control_variate_mc_Qcorr(), NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonDMultilevelSampling::multilevel_mc_Qsum(), NonDMultilevelSampling::multilevel_mc_Ysum(), and NonDMultilevelSampling::NLev.
```cpp
void post_run ( std::ostream & s ) [protected], [virtual]
post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way.

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.
References Analyzer::post_run(), and NonDSampling::update_final_statistics().

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Analyzer.
References NonDMultilevelSampling::equivHFEvals, Analyzer::iteratedModel, NonDMultilevelSampling::NLev, NonDSampling::print_moments(), NonD::print_multilevel_evaluation_summary(), Model::response_labels(), NonDSampling::statsFlag, and Model::truth_model().

void multilevel_mc_Ysum ( size_t model_form ) [private]
Perform multilevel Monte Carlo across the discretization levels for a particular model form using discrepancy accumulators (sum_Y)

This function performs "geometrical" MLMC on a single model form with multiple discretization levels.

References NonDMultilevelSampling::accumulate_ml_Ysums(), NonDMultilevelSampling::aggregate_mse_Ysum(), NonDMultilevelSampling::aggregate_variance_Ysum(), NonDMultilevelSampling::average(), Iterators::convergenceTol, NonDMultilevelSampling::convert_moments(), NonDMultilevelSampling::equivHFEvals, Analyzer::evaluate_parameter_sets(), NonDMultilevelSampling::export_all_samples(), NonDMultilevelSampling::exportSampleSets, NonDSampling::get_parameter_sets(), NonDMultilevelSampling::initialize_ml_Ysums(), Iterators::maxIterations, NonD::load_pilot_sample(), NonD::load_pilot_sample(), NonD::momentStats, NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDSampling::numSamples, NonDMultilevelSampling::one_sided_delta(), Iterators::outputLevel, NonDMultilevelSampling::pilotSamples, Model::solution_level_costs(), Model::solution_levels(), Model::surrogate_model(), Model::surrogate_model_indices(), Model::surrogate_response_mode(), Model::truth_model(), and Model::truth_model_indices().

Referenced by NonDMultilevelSampling::core_run().

void multilevel_mc_Qsum ( size_t model_form ) [private]
Perform multilevel Monte Carlo across the discretization levels for a particular model form using QoI accumulators (sum_Q)

This function performs "geometrical" MLMC on a single model form with multiple discretization levels.

References NonDMultilevelSampling::accumulate_ml_Qsums(), NonDMultilevelSampling::aggregate_mse_Qsum(), NonDMultilevelSampling::aggregate_variance_Qsum(), NonDMultilevelSampling::average(), NonDMultilevelSampling::centered_to_standard(), NonDMultilevelSampling::check_negative(), NonDMultilevelSampling::compute_error_estimates(), Iterators::convergenceTol, NonDMultilevelSampling::equivHFEvals, Analyzer::evaluate_parameter_sets(), NonDMultilevelSampling::export_all_samples(), NonDMultilevelSampling::exportSampleSets, NonD::final_MomentsType, NonDSampling::get_parameter_sets(), NonDMultilevelSampling::initialize_ml_Qsums(), Iterators::iteratedModel, NonD::load_pilot_sample(), Iterators::maxIterations, NonD::momentStats, NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDSampling::numSamples, NonDMultilevelSampling::one_sided_delta(), Iterators::...
::outputLevel, NonDMultilevelSampling::pilotSamples, Model::solution_level_costs(), Model::solution_levels(), Model::surrogate_model(), Model::surrogate_model_indices(), Model::surrogate_response_mode(), Model::truth_model(), Model::truth_model_indices(), and NonDMultilevelSampling::uncentered_to_centered.

Referenced by NonDMultilevelSampling::core_run().

```cpp
void control_variate_mc ( const SizetSizetPair & lf_form_level, const SizetSizetPair & hf_form_level )
{[private]

Perform control variate Monte Carlo across two model forms.

This function performs control variate MC across two combinations of model form and discretization level.

References NonDMultilevelSampling::accumulate_cv_sums(), Iterator::convergenceTol, NonDMultilevelSampling::convert_moments(), NonDMultilevelSampling::cv_raw_moments(), NonDMultilevelSampling::equivHFEvals, NonDMultilevelSampling::eval_ratio(), NonDMultilevelSampling::initialize_cv_sums(), Iterator::iteratedModel, NonDMultilevelSampling::lf_increment(), NonD::load_pilot_sample(), Iterator::maxIterations, NonD::momentStats, NonDMultilevelSampling::MSE_ratio(), NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDMultilevelSampling::numSamples, NonDMultilevelSampling::pilotSamples, NonDMultilevelSampling::shared_increment(), Model::solution_level_cost(), Model::surrogate_model(), Model::surrogate_model_indices(), Model::surrogate_response_mode(), Model::truth_model(), and Model::truth_model_indices().

Referenced by NonDMultilevelSampling::core_run().

void multilevel_control_variate_mc_Ycorr ( size_t lf_model_form, size_t hf_model_form ) [private]

Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for Y (LH correlations for level discrepancies)

This function performs "geometrical" MLMC across discretization levels for the high fidelity model form where CVMC is employed across two model forms to exploit correlation in the discrepancies at each level (Y_l).

References NonDMultilevelSampling::accumulate_ml_Ysums(), NonDMultilevelSampling::accumulate_mlcv_Ysums(), NonDMultilevelSampling::aggregate_mse_Ysum(), NonDMultilevelSampling::aggregate_mse_Yvar(), NonDMultilevelSampling::aggregate_variance_Ysum(), Analyzer::allResponses, NonDMultilevelSampling::average(), Iterator::convergenceTol, NonDMultilevelSampling::convert_moments(), NonDMultilevelSampling::cv_raw_moments(), NonDMultilevelSampling::equivHFEvals, NonDMultilevelSampling::eval_ratio(), Analyzer::evaluate_parameter_sets(), NonDMultilevelSampling::export_all_samples(), NonDMultilevelSampling::exportSampleSets, NonDMultilevelSampling::get_parameter_sets(), NonDMultilevelSampling::initialize_mlcv_sums(), Iterator::iteratedModel, NonDMultilevelSampling::lf_increment(), NonD::load_pilot_sample(), Iterator::maxIterations, NonD::momentStats, NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDMultilevelSampling::numSamples, NonDMultilevelSampling::one_sided_delta(), Iterator::outputLevel, NonDMultilevelSampling::pilotSamples, Model::solution_level_costs(), Model::solution_levels(), NonDMultilevelSampling::sum(), Model::surrogate_model(), Model::surrogate_model_indices(), Model::surrogate_response_mode(), Model::truth_model(), and Model::truth_model_indices().

Referenced by NonDMultilevelSampling::core_run().

void multilevel_control_variate_mc_Qcorr ( size_t lf_model_form, size_t hf_model_form ) [private]

Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for Q (LH correlations for QoI)

This function performs "geometrical" MLMC across discretization levels for the high fidelity model form where CVMC is employed across two model forms. It generalizes the Y_l correlation case to separately target correlations for each QoI level embedded within the level discrepancies.

References NonDMultilevelSampling::accumulate_ml_Ysums(), NonDMultilevelSampling::accumulate_mlcv_Qsums(), NonDMultilevelSampling::aggregate_mse_Ysum(), NonDMultilevelSampling::aggregate_mse_Yvar(), NonDMultilevelSampling::aggregate_variance_Ysum(), Analyzer::allResponses, NonDMultilevelSampling::average(),
void uncentered_to_centered ( Real \textit{rm1}, Real \textit{rm2}, Real \textit{rm3}, Real \textit{rm4}, Real \& \textit{cm1}, Real \& \textit{cm2}, Real \& \textit{cm3}, Real \& \textit{cm4}, \textit{size1 \_Nlq} ) \textbf{const [inline], [private]}

convert uncentered (raw) moments to centered moments; biased estimators
For single-level moment calculations with a scalar \textit{Nlq}.
Referenced by NonDMultilevelSampling::compute_error_estimates(), NonDMultilevelSampling::convert_moments(), and NonDMultilevelSampling::multilevel_mc_Qsum().

void uncentered_to_centered ( Real \textit{rm1}, Real \textit{rm2}, Real \textit{rm3}, Real \textit{rm4}, Real \& \textit{cm1}, Real \& \textit{cm2}, Real \& \textit{cm3}, Real \& \textit{cm4} ) \textbf{const [inline], [private]}

convert uncentered (raw) moments to centered moments; unbiased estimators
For single-level moment calculations with a scalar \textit{Nlq}.
The documentation for this class was generated from the following files:

- NonDMultilevelSampling.hpp
- NonDMultilevelSampling.cpp

13.122 NonDMultilevelStochCollocation Class Reference
Nonintrusive stochastic collocation approaches to uncertainty quantification.
Inheritance diagram for NonDMultilevelStochCollocation:

```
  NonDMultilevelStochCollocation
     |     |
     |     | NonD
     |     |
     |     | NonDExpansion
     |     |
     |     | NonDStochCollocation
     |     |
  Iterator
```

Referenced by NonDMultilevelSampling::core_run().
Public Member Functions

- NonDMultilevelStochCollocation (ProblemDescDB &problem_db, Model &model)
  standard constructor
- NonDMultilevelStochCollocation (Model &model, short exp_coeffs_approach, const UShortArray &num_int_seq, const RealVector &dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs)
  alternate constructor
- ~NonDMultilevelStochCollocation ()
  destructor
- bool resize ()
  reinitializes iterator based on new variable size

Protected Member Functions

- void core_run ()
  perform a forward uncertainty propagation using PCE/SC methods
- void increment_specification_sequence ()
  increment the input specification sequence (PCE only)

Private Attributes

- UShortArray quadOrderSeqSpec
  user request of quadrature order
- UShortArray ssgLevelSeqSpec
  user request of sparse grid level
- size_t sequenceIndex
  sequence index for quadOrderSeqSpec and ssgLevelSeqSpec

Additional Inherited Members

13.122.1 Detailed Description

Nonintrusive stochastic collocation approaches to uncertainty quantification.

The NonDMultilevelStochCollocation class uses a stochastic collocation (SC) approach to approximate the
effect of parameter uncertainties on response functions of interest. It utilizes the InterpPolyApproximation class
to manage multidimensional Lagrange polynomial interpolants.

13.122.2 Constructor & Destructor Documentation

NonDMultilevelStochCollocation ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.
References Response::active_set(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(),
ParallelLibrary::command_line_check(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), NonDExpansion::construct_expansion_sampler(), Model::current_response(), ProblemDescDB::get_bool(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), NonDExpansion::initialize_random(), NonDStochCollocation::initialize_u_space_model(),

NonDMultilevelStochCollocation ( Model & model, short exp_coeffs_approach, const UShortArray & num_int_seq, const RealVector & dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs )

alternate constructor

This constructor is used for helper iterator instantiation on the fly.

References Response::active_set(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), Model::current_response(), NonDExpansion::expansionCoeffsApproach, NonDExpansion::initialize_random(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, NonDMultilevelStochCollocation::quadOrderSeqSpec, ActiveSet::request_values(), NonDStochCollocation::resolve_inputs(), NonDMultilevelStochCollocation::sequenceIndex, NonDMultilevelStochCollocation::ssgLevelSeqSpec, NonD::transform_model(), and NonDExpansion::uSpaceModel.

The documentation for this class was generated from the following files:

- NonDMultilevelStochCollocation.hpp
- NonDMultilevelStochCollocation.cpp

13.123 NonDPOFDarts Class Reference

Base class for POF Dart methods within DAKOTA/UQ.

Inheritance diagram for NonDPOFDarts:

```
NonDPOFDarts
     |      |
     v      v
NonD
     |
     v
Analyzer
     |
     v
Iterator
```

Public Member Functions

- **NonDPOFDarts (ProblemDescDB &problem_db, Model &model)**
  
  *constructor*

- **~NonDPOFDarts ()**
  
  *destructor*

- **bool resize ()**
  
  *reinitializes iterator based on new variable size*

- **void core_run ()**
  
  *perform POFDart analysis and return probability of failure*
**Protected Member Functions**

- void initiate_random_number_generator (unsigned long x)

  *POF Darts Methods.*

- double generate_a_random_number ()
- void init_pof_darts ()
- void exit_pof_darts ()
- void execute (size_t kd)
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  
  *print the final statistics*

- void classical_dart_throwing_games (size_t game_index)
- void line_dart_throwing_games (size_t game_index)
- bool valid_dart (double *x)
- bool valid_line_flat (size_t flat_dim, double *flat_dart)
- void add_point (double *x)
- void compute_response (double *x)
- void verify_neighbor_consistency ()
- bool add_neighbor (size_t ipoint, size_t inneighbor)
- void retrieve_neighbors (size_t ipoint, bool update_point_neighbors)
- void sample_furthest_vertex (size_t ipoint, double *fv)
- void update_global_L ()
- void assign_sphere_radius_POF (size_t isample)
- void shrink_big_spheres ()
- double area_triangle (double x1, double y1, double x2, double y2, double x3, double y3)
- void initialize_surrogates ()
- void add_surrogate_data (const Variables &vars, const Response &resp)
- void build_surrogate ()
- double eval_surrogate (size_t fn_index, double *vin)
- void estimate_pof_surrogate ()
- bool trim_line_using_Hyperplane (size_t num_dim, double *st, double *end, double *qH, double *nH)
- double f_true (double *x)
- void plot_vertices_2d (bool plot_true_function, bool plot_surrogate)
- void plot_neighbors ()

**Protected Attributes**

- int samples
- int seed
- int emulatorSamples
- String lipschitzType
- RealRealPairArray extremeValues
- double Q [1220]
- int indx
- double cc
- double c
- double zc
- double zx
• double zy
• size_t qlen
• bool _eval_error
• size_t _test_function
• size_t _n_dim
• double * _xmin
• double * _xmax
• double _diag
• double _failure_threshold
• double _num_darts
• double _num_successive_misses_p
• double _num_successive_misses_m
• double _max_num_successive_misses
• double _accepted_void_ratio
• size_t _num_inserted_points
• size_t _total_budget
• double *** _sample_points
• size_t *** _sample_neighbors
• double * _sample_vsize
• double _max_vsize
• double * _dart
• size_t _flat_dim
• size_t * _line_flat
• size_t _num_flat_segments
• double * _line_flat_start
• double * _line_flat_end
• double * _line_flat_length
• double _safety_factor
• double * _Lip
• double *** _fval
• size_t _active_response_function
• bool _use_local_L

Additional Inherited Members

13.123.1 Detailed Description

Base class for POF Dart methods within DAKOTA/UQ.

The NonDPOFDart class implements the calculation of a failure probability for a specified threshold for a specified response function using the concepts developed by Mohamed Ebeida. The approach works by throwing down a number of Poisson disk samples of varying radii, and identifying each disk as either in the failure or safe region. The center of each disk represents a "true" function evaluation. kd-darts are used to place additional points, in such a way to target the failure region. When the disks cover the space sufficiently, Monte Carlo methods or a box volume approach is used to calculate both the lower and upper bounds on the failure probability.

The documentation for this class was generated from the following files:

• NonDPOFDarts.hpp
• NonDPOFDarts.cpp
13.124 NonDPolynomialChaos Class Reference

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

Inheritance diagram for NonDPolynomialChaos:

```
Public Member Functions

- NonDPolynomialChaos (ProblemDescDB &problem_db, Model &model)
  standard constructor
- NonDPolynomialChaos (Model &model, short exp_coeffs_approach, unsigned short num_int, const RealVector &dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs)
  alternate constructor for numerical integration (tensor, sparse, cubature)
- NonDPolynomialChaos (Model &model, short exp_coeffs_approach, unsigned short exp_order, const RealVector &dim_pref, size_t colloc_pts, Real colloc_ratio, int seed, short u_space_type, bool piecewise_basis, bool use_derivs, bool cv_flag, const String &import_build_pts_file, unsigned short import_build_format, bool import_build_active_only)
  alternate constructor for regression (least squares, CS, OLI)
- ~NonDPolynomialChaos ()
  destructor
- bool resize ()
  reinitializes iterator based on new variable size

Protected Member Functions

- NonDPolynomialChaos (BaseConstructor, ProblemDescDB &problem_db, Model &model)
  base constructor for DB construction of multilevel/multifidelity PCE
- NonDPolynomialChaos (unsigned short method_name, Model &model, short exp_coeffs_approach, const RealVector &dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs)
  base constructor for lightweight construction of multilevel/multifidelity PCE using numerical integration
- NonDPolynomialChaos (unsigned short method_name, Model &model, short exp_coeffs_approach, const RealVector &dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs, Real colloc_ratio, int seed, bool cv_flag)
base constructor for lightweight construction of multilevel/multifidelity PCE using regression

- void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance

- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance

- void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance

- void resolve_inputs (short &u_space_type, short &data_order)
  perform error checks and mode overrides

- void initialize_u_space_model ()
  initialize uSpaceModel polynomial approximations with PCE/SC data

- void compute_expansion (size_t index=_NPOS)
  form or import an orthogonal polynomial expansion using PCE methods

- void select_refinement_points (const RealVectorArray &candidate_samples, unsigned short batch_size, RealMatrix &best_samples)
  evaluate allSamples for inclusion in the (PCE regression) approximation and retain the best set (well spaced) of size batch_size

- void select_refinement_pointsDeprecated (const RealVectorArray &candidate_samples, unsigned short batch_size, RealMatrix &best_samples)

- void append_expansion (const RealMatrix &samples, const IntResponseMap &resp_map, size_t index=_NPOS)
  append new data to uSpaceModel and update expansion order (PCE only)

- void increment_order_and_grid ()

- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final coefficients and final statistics

- void print_coefficients (std::ostream &s)
  print the PCE coefficient array for the orthogonal basis

- void export_coefficients ()
  export the PCE coefficient array to expansionExportFile

- void archive_coefficients ()
  archive the PCE coefficient array for the orthogonal basis

- void append_expansion (size_t index=_NPOS)
  generate new samples from numSamplesOnModel and update expansion

- void config_expansion_orders (unsigned short exp_order, const RealVector &dim_pref, UShortArray &exp_orders)
  configure exp_orders from inputs

- bool config_integration (unsigned short quad_order, unsigned short ssg_level, unsigned short cub_int, Iterator &u_space_sampler, Model &g_u_model, String &approx_type)
  configure u_space_sampler and approx_type based on numerical integration specification

- bool config_expectation (size_t exp_samples, unsigned short sample_type, const String &rng, Iterator &u_space_sampler, Model &g_u_model, String &approx_type)
  configure u_space_sampler and approx_type based on expansion_samples specification
• bool config_regression (const UShortArray &exp_orders, size_t colloc_pts, Real colloc_ratio, order, short regress_type, short ls_regress_type, const UShortArray &tensor_grid_order, unsigned short sample_type, const String &rng, const String &pt_reuse, Iterator &u_space_sampler, Model &g_u_model, String &approx_type)

configure u_space_sampler and approx_type based on regression specification

• int terms_ratio_to_samples (size_t num_exp_terms, Real colloc_ratio)

calculate number of expansion terms and collocation ratio to a number of collocation samples

• Real terms_samples_to_ratio (size_t num_exp_terms, int samples)

convert number of expansion terms and number of collocation samples to a collocation ratio

• void ratio_samples_to_order (Real colloc_ratio, int num_samples, UShortArray &exp_order, bool less_than_or_equal)

convert collocation ratio and number of samples to expansion order

Protected Attributes

• short uSpaceType

user requested expansion type

• unsigned short cubIntSpec

cubature integrand

• RealVector dimPrefSpec

user specification for dimension_preference

• Real collocRatio

factor applied to terms^termsOrder in computing number of regression points, either user specified or inferred

• bool tensorRegression

option for regression PCE using a filtered set tensor-product points

• String importBuildPointsFile

user specified import build points file

• unsigned short importBuildFormat

user specified import build file format

• bool importBuildActiveOnly

user specified import build active only

• String importApproxPointsFile

user specified import approx. points file

• unsigned short importApproxFormat

user specified import approx. file format

• bool importApproxActiveOnly

user specified import approx. active only

• String expansionImportFile

filename for import of chaos coefficients

• String expansionExportFile

filename for export of chaos coefficients
CHAPTER 13. CLASS DOCUMENTATION

Private Member Functions

- void increment_grid_from_order ()
  define a grid increment that is consistent with an advancement in expansion order
- void increment_order_from_grid ()
  define an expansion order that is consistent with an advancement in structured/unstructured grid level/density
- void order_to_dim_preference (const UShortArray &order, unsigned short &p, RealVector &dim_pref)
  convert an isotropic/anisotropic expansion order vector into a scalar plus a dimension preference vector

Private Attributes

- Real termsOrder
  exponent applied to number of expansion terms for computing number of regression points
- int randomSeed
  seed for random number generator used for regression with LHS and sub-sampled tensor grids
- bool crossValidation
  flag for use of cross-validation for selection of parameter settings in regression approaches
- bool crossValidNoiseOnly
  flag to restrict cross-validation to only estimate the noise tolerance in order to manage computational cost
- RealVector noiseTols
  noise tolerance for compressive sensing algorithms; vector form used in cross-validation
- Real l2Penalty
  L2 penalty for LASSO algorithm (elastic net variant)
- unsigned short numAdvance
  number of frontier expansions per iteration with the ADAPTED BASIS EXPANDING FRONT approach
- unsigned short expOrderSpec
  user specification for expansion order (array for multifidelity)
- size_t collocPtsSpec
  user specification for collocation points (array for multifidelity)
- size_t expSamplesSpec
  user specification for expansion samples (array for multifidelity)
- unsigned short quadOrderSpec
  user request of quadrature order
- unsigned short ssgLevelSpec
  user request of sparse grid level
- RealMatrix pceGradsMeanX
  derivative of the PCE with respect to the x-space variables evaluated at the means (used as uncertainty importance metrics)
- bool normalizedCoeffOutput
  user request for use of normalization when outputting PCE coefficients
Additional Inherited Members

13.124.1 Detailed Description

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

The NonDPolynomialChaos class uses a polynomial chaos expansion (PCE) approach to approximate the
effect of parameter uncertainties on response functions of interest. It utilizes the OrthogPolyApproximation class
to manage multiple types of orthogonal polynomials within a Wiener-Askey scheme to PCE. It supports PCE
coefficient estimation via sampling, quadrature, point-collocation, and file import.

13.124.2 Constructor & Destructor Documentation

NonDPolynomialChaos ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Dakota::abort_handler(), Response::active_set(), Model::assign_rep(), NonDExpansion::check_-
dimension_preference(), NonDPolynomialChaos::collocPtsSpec, ParallelLibrary::command_line_check(), NonD-
PolynomialChaos::config_expansion_orders(), NonDPolynomialChaos::config_expectation(), NonDPolynomial-
Chaos::config_integration(), NonDPolynomialChaos::config_regression(), NonDExpansion::construct_expansion-
_sampler(), NonDPolynomialChaos::cubIntSpec, Model::current_response(), NonDPolynomialChaos::dimPrefSpec,
NonDPolynomialChaos::expansionImportFile, NonDPolynomialChaos::expOrderSpec, NonDPolynomialChaos-
::expSamplesSpec, ProblemDescDB::get_real(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), Problem-
DescDB::get_usa(), NonDPolynomialChaos::importApproxActiveOnly, NonDPolynomial-
Chaos::importApproxFormat, NonDPolynomialChaos::importApproxPointsFile, NonDPolynomialChaos::import-
BuildActiveOnly, NonDPolynomialChaos::importBuildFormat, NonDPolynomialChaos::importBuildPointsFile,
NonDExpansion::initialize_random(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel,
NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, Iterator::probDescDB, Non-
DPolynomialChaos::quadOrderSpec, ActiveSet::request_values(), NonDPolynomialChaos::resolve_inputs(), Non-
DPolynomialChaos::ssgLevelSpec, NonD::transform_model(), NonDExpansion::uSpaceModel, and NonDPolynomial-
Chaos::uSpaceType.

NonDPolynomialChaos ( Model & model, short exp_coeffs_approach, unsigned short num_int, const
RealVector & dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs )

alternate constructor for numerical integration (tensor, sparse, grid, curvature)

This constructor is used for helper iterator instantiation on the fly that employ numerical integration (quadrature,
sparse grid, curvature).

References Dakota::abort_handler(), Response::active_set(), Model::assign_rep(), NonDExpansion::check_-
dimension_preference(), NonDPolynomialChaos::config_integration(), Model::current_response(), NonDPolynomial-
Chaos::dimPrefSpec, NonDExpansion::initialize_random(), NonDPolynomialChaos::initialize_u_space_model(),
Iterator::iteratedModel, Iterator::outputLevel, ActiveSet::request_values(), NonDPolynomialChaos::resolve_inputs(),
NonD::transform_model(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDPolynomialChaos ( Model & model, short exp_coeffs_approach, unsigned short exp_order, const RealVector & dim_pref, size_t colloc_pts, Real colloc_ratio, int seed, short u_space_type, bool piecewise_basis, bool use_derivs, bool cv_flag, const String & import_build_pts_file, unsigned short import_build_format, bool import_build_active_only )

alternate constructor for regression (least squares, CS, OLI)
CHAPTER 13. CLASS DOCUMENTATION

This constructor is used for helper iterator instantiation on the fly that employ regression (least squares, CS, OLI).

References Response::active_set(), Model::assign_rep(), NonDExpansion::check_dimension_preference(), NonDPolynomialChaos::collocPtsSpec, NonDPolynomialChaos::config_expansion_orders(), NonDPolynomialChaos::config_regression(), Model::current_response(), NonDPolynomialChaos::dimPrefSpec, NonDPolynomialChaos::expOrderSpec, NonDExpansion::initialize_random(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, ActiveSet::request_values(), NonDPolynomialChaos::resolve_inputs(), NonD::transform_model(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDPolynomialChaos ( BaseConstructor , ProblemDescDB & problem_db, Model & model ) [protected]

base constructor for DB construction of multilevel/multifidelity PCE

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References NonDExpansion::check_dimension_preference(), and NonDPolynomialChaos::dimPrefSpec.

13.124.3 Member Function Documentation

void increment_order_and_grid ( ) [protected], [virtual]

Used for uniform refinement of regression-based PCE.

Reimplemented from NonDExpansion.

References SharedApproxData::data_rep(), NonDPolynomialChaos::increment_grid_from_order(), SharedPecosApproxData::increment_order(), Model::shared_approximation(), and NonDExpansion::uSpaceModel.

void increment_grid_from_order ( ) [private]

define a grid increment that is consistent with an advancement in expansion order

Used for uniform refinement of regression-based PCE.

References NonDPolynomialChaos::collocRatio, SharedApproxData::data_rep(), SharedPecosApproxData::expansion_order(), NonDExpansion::expansionBasisType, NonDQuadrature::increment_grid(), Iterator::iterator_rep(), NonDQuadrature::model(), Model::model_rep(), NonDExpansion::numSamplesOnModel, NonDQuadrature::samples(), Iterator::sampling_reference(), Model::shared_approximation(), Model::subordinate_iterator(), NonDPolynomialChaos::tensorRegression, NonDPolynomialChaos::terms_ratio_to_samples(), DataFitSurrModel::total_points(), NonDQuadrature::update(), and NonDExpansion::uSpaceModel.

Referenced by NonDPolynomialChaos::increment_order_and_grid().

void increment_order_from_grid ( ) [private]

define an expansion order that is consistent with an advancement in structured/unstructured grid level/density

Used for uniform refinement of regression-based PCE.

References NonDPolynomialChaos::collocRatio, SharedApproxData::data_rep(), SharedPecosApproxData::expansion_order(), NonDExpansion::numSamplesOnModel, NonDPolynomialChaos::ratio_samples_to_order(), Model::shared_approximation(), and NonDExpansion::uSpaceModel.

Referenced by NonDPolynomialChaos::append_expansion().

The documentation for this class was generated from the following files:

- NonDPolynomialChaos.hpp
- NonDPolynomialChaos.cpp
13.125  **NonDQuadrature Class Reference**

Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

Inheritance diagram for NonDQuadrature:

![Inheritance Diagram](image)

**Public Member Functions**

- **NonDQuadrature (Model &model, unsigned short quad_order, const RealVector &dim_pref, short driver_mode)**
  
  alternate constructor for instantiations "on the fly" based on a quadrature order specification

- **NonDQuadrature (Model &model, unsigned short quad_order, const RealVector &dim_pref, short driver_mode, int num_filt_samples)**
  
  alternate constructor for instantiations "on the fly" that filter a tensor product sample set to include points with highest sample weights

- **NonDQuadrature (Model &model, unsigned short quad_order, const RealVector &dim_pref, short driver_mode, int num_sub_samples, int seed)**
  
  alternate constructor for instantiations "on the fly" that sub-sample quadrature rules by sampling randomly from a tensor product multi-index

- **void increment_grid ()**
  
  increment SSG level/TPQ order

- **void update ()**
  
  propagate any numSamples updates and/or grid updates/increments

- **void reset ()**
  
  set dimQuadOrderRef to dimension orders indicated by quadOrderSpec and dimPrefSpec, following refinement or sequence advancement

- **const Pecos::UShortArray & quadrature_order () const**

  return Pecos::TensorProductDriver::quadOrder

- **void quadrature_order (const Pecos::UShortArray &dim_quad_order)**

  set dimQuadOrderRef and map to Pecos::TensorProductDriver::quadOrder

- **void quadrature_order (unsigned short quad_order)**

  set quadOrderSpec and map to Pecos::TensorProductDriver::quadOrder

- **void samples (size_t samples)**
CHAPTER 13. CLASS DOCUMENTATION

set numSamples

- short mode () const
  return quadMode

Protected Member Functions

- NonDQuadrature (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDQuadrature ()
  destructor
- void initialize_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)
- void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.

- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
- void increment_grid_preference (const RealVector &dim_pref)
  increment SSG level/TPQ order and update anisotropy

- int num_samples () const

Private Member Functions

- void increment_grid (UShortArray &dim_quad_order)
  convenience function used to make increment_grid() more modular
- void increment_grid_preference (const RealVector &dim_pref, UShortArray &dim_quad_order)
  convenience function used to make increment_grid_preference() more modular

- void compute_minimum_quadrature_order (size_t min_samples, const RealVector &dim_pref, UShortArray &dim_quad_order)
  calculate smallest dim_quad_order with at least min_samples

- void filter_parameter_sets ()
  prune allSamples back to size numSamples, retaining points with highest product weight

- void update_anisotropic_order (const RealVector &dim_pref, UShortArray &quad_order_ref)
  update quad_order_ref based on an updated dimension preference, enforcing previous values as a lower bound

- void initialize_dimension_quadrature_order (unsigned short quad_order_spec, const RealVector &dim_pref_spec, UShortArray &dim_quad_order)
  initialize dim_quad_order from quad_order_spec and dim_pref_spec

- void increment_dimension_quadrature_order (UShortArray &dim_quad_order)
  increment each dim_quad_order entry by 1

- void increment_dimension_quadrature_order (const RealVector &dim_pref, UShortArray &dim_quad_order)
  increment the dim_quad_order entry with maximum preference by 1 and then rebalance
Private Attributes

- Pecos::TensorProductDriver * tpqDriver
  
  convenience pointer to the numIntDriver representation

- bool nestedRules
  
  for studies involving refinement strategies, allow for use of nested quadrature rules such as Gauss-Patterson

- unsigned short quadOrderSpec
  
  scalar quadrature order, rendered anisotropic via dimPrefSpec

- UShortArray dimQuadOrderRef
  
  reference point for Pecos::TensorProductDriver::quadOrder: the original user specification for the number of
  Gauss points per dimension, plus any refinements posted by increment_grid()

- short quadMode
  
  point generation mode: FULL_TENSOR, FILTERED_TENSOR, RANDOM_TENSOR

- size_t numSamples
  
  size of a subset of tensor quadrature points (filtered based on product weight or sampled uniformly from the tensor
  multi-index); used by the regression PCE approach known as "probabilistic collocation"

- int randomSeed
  
  seed for the random number generator used in sampling of the tensor multi-index

Additional Inherited Members

13.125.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expec-
tation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of
moments. It employs Gauss-Hermite, Gauss-Legendre, Gauss-Laguerre, Gauss-Jacobi and generalized Gauss--
Laguerre quadrature for use with normal, uniform, exponential, beta, and gamma density functions and integration
bounds. The abscissas and weights for one-dimensional integration are extracted from the appropriate Orthogonal-
Polynomial class and are extended to n-dimensions using a tensor product approach.

13.125.2 Constructor & Destructor Documentation

NonDQuadrature ( Model & model, unsigned short quad_order, const RealVector & dim_pref, short
driver_mode )

alternate constructor for instantiations “on the fly” based on a quadrature order specification

This alternate constructor is used for on-the-fly generation and evaluation of numerical quadrature points.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

NonDQuadrature ( Model & model, unsigned short quad_order, const RealVector & dim_pref, short
driver_mode, int num_filt_samples )

alternate constructor for instantiations “on the fly” that filter a tensor product sample set to include points with
highest sample weights

This alternate constructor is used for on-the-fly generation and evaluation of filtered tensor quadrature points.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.
NonDQuadrature ( Model & model, unsigned short quad_order, const RealVector & dim_pref, short driver_mode, int num_sub_samples, int seed )

alternate constructor for instantiations "on the fly" that sub-sample quadrature rules by sampling randomly from a tensor product multi-index

This alternate constructor is used for on-the-fly generation and evaluation of random sampling from a tensor quadrature multi-index.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

NonDQuadrature ( ProblemDescDB & problem_db, Model & model ) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate nond_quadrature method specification.

References NonDIntegration::check_variables(), Iterator::convergenceTol, Model::correction_type(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_short(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, NonD::natafTransform, NonDQuadrature::nestedRules, NonDIntegration::numIntDriver, Iterator::outputLevel, Iterator::probDescDB, NonDQuadrature::reset(), and NonDQuadrature::tpqDriver.

13.125.3 Member Function Documentation

void initialize_grid ( const std::vector<Pecos::BasisPolynomial> & poly_basis ) [protected], [virtual]

Used in combination with alternate NonDQuadrature constructor.

Implements NonDIntegration.

References Iterator::maxEvalConcurrency, NonDQuadrature::nestedRules, Analyzer::numContinuousVars, NonDQuadrature::numSamples, NonDQuadrature::quadMode, NonDQuadrature::reset(), NonDQuadrature::tpqDriver, and NonDQuadrature::update().

void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the quadrature routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDQuadrature::compute_minimum_quadrature_order(), NonDIntegration::dimPrefSpec, NonDQuadrature::dimQuadOrderRef, NonDQuadrature::nestedRules, Analyzer::numContinuousVars, and NonDQuadrature::tpqDriver.

Referenced by NonDQuadrature::update().

int num_samples ( ) const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References NonDQuadrature::numSamples, NonDQuadrature::quadMode, and NonDQuadrature::tpqDriver.

The documentation for this class was generated from the following files:

- NonDQuadrature.hpp
- NonDQuadrature.cpp
13.126 NonDQUESOBayesCalibration Class Reference

Bayesian inference using the QUESO library from UT Austin.

Inheritance diagram for NonDQUESOBayesCalibration:

```
  Iterator
   |   Analyzer
   |   NonD
   | -- NonDCalibration
       | -- NonDBayesCalibration
           | -- NonDQUESOBayesCalibration
```

### Public Member Functions

- **NonDQUESOBayesCalibration (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **~NonDQUESOBayesCalibration ()**
  
  *destructor*

### Protected Member Functions

- void **calibrate ()**
- void **print_results (std::ostream &s, short results_state=FINAL_RESULTS)**
  
  *print the final iterator results*

- void **init_queso_environment ()**
  
  *initialize the QUESO FullEnvironment on the Dakota MPIComm*

- void **init_precond_request_value ()**
  
  *initialize the ASV value for preconditioned cases*

- virtual void **init_queso_solver ()**
  
  *define solver options, likelihood callback, posterior RV, and inverse problem*

- void **precondition_proposal ()**
  
  *use derivative information from the emulator to define the proposal covariance (inverse of misfit Hessian)*

- void **run_queso_solver ()**
  
  *perform the MCMC process*

- void **run_chain_with_restarting ()**
short term option to restart the MCMC chain with updated proposal density computed from the emulator at a new starting point

- void accumulate\_chain (size\_t update\_cntr)
  accumulate unique samples drawn from the acceptance chain

- void aggregate\_acceptance\_chain (size\_t update\_cntr)
  accumulate the acceptance chain across multiple restart cycles, including recovering corresponding function values

- void filter\_chain\_by\_probability (size\_t update\_cntr, unsigned short batch\_size)
  extract batch\_size points from the MCMC chain and store final aggregated set within all\_Samples; unique points with highest posterior probability are selected

- void filter\_chain\_by\_conditioning (size\_t update\_cntr, unsigned short batch\_size)
  extract batch\_size points from the MCMC chain and store final aggregated set within all\_Samples; unique points with best conditioning are selected, as determined by pivoted LU

- void chain\_to\_local (unsigned short batch\_size, std::map<Real, size\_t> &local\_best)
  store indices of best batch\_size samples from the current MCMC chain within the local\_best array

- void local\_to\_aggregated (unsigned short batch\_size, const std::map<Real, size\_t> &local\_best)
  update best\_Samples aggregation using new contributions from the current MCMC chain

- void aggregated\_to\_all ()
  following aggregation cycles, copy best\_Samples to all\_Samples

- void local\_to\_all (const std::map<Real, size\_t> &local\_best)
  in the absence of aggregation cycles, copy local\_best to all\_Samples

- void update\_center ()
  update the starting point for a restarted MCMC chain using last point from previous chain

- void update\_model ()
  evaluates all\_Samples on iterated\_Model and update the mcmc\_Model emulator with all\{Samples,Responses\}

- Real assess\_emulator\_convergence ()
  compute the L2 norm of the change in emulator coefficients

- void init\_parameter\_domain ()
  initialize the QUESO parameter space, min, max, initial, and domain

- void init\_proposal\_covariance ()

- void prior\_proposal\_covariance ()
  use covariance of prior distribution for setting proposal covariance

- void user\_proposal\_covariance (const String &input\_fmt, const RealVector &cov\_data, const String &cov\_filename)
  set proposal covariance from user-provided diagonal or matrix

- void validate\_proposal ()

- void set\_ip\_options ()
  set inverse problem options call\_Ip\_Options\_Values common to all solvers

- void set\_mh\_options ()
  set MH-specific inverse problem options call\_IpMh\_Options\_Values

- void update\_chain\_size (unsigned int size)
  update MH-specific inverse problem options call\_IpMh\_Options\_Values

- void copy\_gsl (const QUESO::GslVector &qv, RealVector &rv)
  local copy gsl utility from Gsl\_Vector to Real\_Vector

- void copy\_gsl (const RealVector &rv, QUESO::GslVector &qv)
local copy data utility from RealVector to GslVector

- void copy_gsl_partial (const QUESO::GslVector &qv, size_t start, RealVector &rv)
  local copy data utility from portion of GslVector to RealVector
- void copy_gsl_partial (const RealVector &rv, QUESO::GslVector &qv, size_t start)
  local copy data utility from RealVector to portion of GslVector
- void copy_gsl (const QUESO::GslVector &qv, RealMatrix &rm, int i)
  local copy data utility from GslVector to column in RealMatrix
- bool equal_gsl (const QUESO::GslVector &qv1, const QUESO::GslVector &qv2)
  equality tester for two GslVectors

Static Protected Member Functions

- static double dakotaLogLikelihood (const QUESO::GslVector &paramValues, const QUESO::GslVector *paramDirection, const void *functionDataPtr, QUESO::GslVector *gradVector, QUESO::GslMatrix *hessianMatrix, QUESO::GslVector *hessianEffect)
  Log Likelihood function for call-back from QUESO to DAKOTA for evaluation.

Protected Attributes

- String mcmcType
  MCMC type ("dram" or "delayed_rejection" or "adaptive_metropolis" or "metropolis_hastings" or "multilevel", within QUESO)
- short precondRequestValue
  the active set request value to use in proposal preconditioning
- bool logitTransform
  flag indicating user activation of logit transform option
- boost::shared_ptr < QUESO::EnvOptionsValues > envOptionsValues
  options for setting up the QUESO Environment
- boost::shared_ptr < QUESO::FullEnvironment > quesoEnv
  top-level QUESO Environment
- boost::shared_ptr < QUESO::VectorSpace
  < QUESO::GslVector, QUESO::GslMatrix > > paramSpace
  QUESO parameter space based on number of calibrated parameters.
- boost::shared_ptr < QUESO::BoxSubset
  < QUESO::GslVector, QUESO::GslMatrix > > paramDomain
  QUESO parameter domain: hypercube based on min/max values.
- boost::shared_ptr < QUESO::GslVector > paramInitials
  initial parameter values at which to start chain
CHAPTER 13. CLASS DOCUMENTATION

• boost::shared_ptr
  < QUESO::BaseVectorRV
  < QUESO::GslVector,
  QUESO::GslMatrix > > priorRv

• boost::shared_ptr
  < QUESO::GslMatrix > proposalCovMatrix
  proposal covariance for DRAM

• boost::shared_ptr
  < QUESO::SipOptionsValues > callIpOptionsValues
  general inverse problem options

• boost::shared_ptr
  < QUESO::MhOptionsValues > callIpMhOptionsValues
  MH-specific inverse problem options.

• boost::shared_ptr
  < QUESO::GenericScalarFunction
  < QUESO::GslVector,
  QUESO::GslMatrix > > likelihoodFunctionObj

• boost::shared_ptr
  < QUESO::GenericVectorRV
  < QUESO::GslVector,
  QUESO::GslMatrix > > postRv

• boost::shared_ptr
  < QUESO::StatisticalInverseProblem
  < QUESO::GslVector,
  QUESO::GslMatrix > > inverseProb

• String advancedOptionsFile
  advanced options file name (GPMSA only); settings from this file override any C++ / Dakota input file settings

Static Protected Attributes

• static NonDQUESOBayesCalibration * nonDQUESOInstance
  Pointer to current class instance for use in static callback functions.

Private Attributes

• RealVectorArray uniqueSamples
  container for aggregating unique MCMC sample points collected across multiple (restarted) chains

• RealVectorArray prevCoeffs
  cache previous expansion coefficients for assessing convergence of emulator refinement process

• RealVector mapSoln
  store MAP estimate for warm-starting pre-solves

Additional Inherited Members

13.126.1 Detailed Description

Bayesian inference using the QUESO library from UT Austin.
This class provides a wrapper to the QUESO library developed as part of the Predictive Science Academic Alliance Program (PSAAP), specifically the PECOS (Predictive Engineering and Computational Sciences) Center at UT Austin. The name QUESO stands for Quantification of Uncertainty for Estimation, Simulation, and Optimization.

### 13.126.2 Constructor & Destructor Documentation

**NonDQUESOBayesCalibration** (ProblemDescDB & *problem_db*, Model & *model*)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDBayesCalibration::adaptPosteriorRefine, NonDQUESOBayesCalibration::advancedOptionsFile, NonDCalibration::calibrationData, NonDBayesCalibration::chainCycles, NonDBayesCalibration::chainSamples, NonDBayesCalibration::emulatorType, ProblemDescDB::get_int(), NonDQUESOBayesCalibration::init_queso_environment(), Iterator::maxIterations, NonDBayesCalibration::obsErrorMultiplierMode, Iterator::outputLevel, Iterator::probDescDB, and NonDBayesCalibration::proposalCovarType.

### 13.126.3 Member Function Documentation

**void calibrate** ( ) [protected], [virtual]

Perform the uncertainty quantification

- Implements NonDBayesCalibration.

  References Dakota::abort_handler(), NonDBayesCalibration::adaptPosteriorRefine, NonDQUESOBayesCalibration::assess_emulator_convergence(), Analyzer::compactMode, NonDBayesCalibration::compute_statistics(), Iterator::convergenceTol, NonDBayesCalibration::emulatorType, NonDQUESOBayesCalibration::init_parameter_domain(), NonDQUESOBayesCalibration::init_precond_request_value(), NonDQUESOBayesCalibration::init_queso_solver(), NonDBayesCalibration::initialize_model(), Iterator::maxIterations, NonDQUESOBayesCalibration::nonDQUESOInstance, NonDQUESOBayesCalibration::precondRequestValue, NonDBayesCalibration::proposalCovarType, NonDQUESOBayesCalibration::run_chain_with_restarting(), and NonDQUESOBayesCalibration::update_model().

**void print_results** (std::ostream & *s*, short *results_state* = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

  Reimplemented from NonDBayesCalibration.

  References NonDBayesCalibration::bestSamples, Model::continuous_variable_labels(), Dakota::copy_data(), NonDQUESOBayesCalibration::copy_gsl(), NonDQUESOBayesCalibration::copy_gsl_partial(), NonDCalibration::expData, Dakota::HALF_LOG_2PI, ExperimentData::half_log_cov_determinant(), Iterator::iteratedModel, NonDBayesCalibration::log_prior_density(), NonD::natafTransform, Model::num_primary_fns(), Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, NonDQUESOBayesCalibration::paramSpace, NonDBayesCalibration::print_results(), NonDBayesCalibration::residualModel, NonDBayesCalibration::standardizedSpace, and Dakota::write_precision.

**void aggregate_acceptance_chain** (size_t *cycle_num*) [protected]

accumulate the acceptance chain across multiple restart cycles, including recovering corresponding function values
Populate a subset of acceptanceChain(num_params, chainSamples * chainCycles) and acceptedFnVals(num-Functions, chainSamples * chainCycles)

References NonDBayesCalibration::acceptanceChain, NonDBayesCalibration::acceptedFnVals, Response::active- set(), Model::active_variables(), NonDBayesCalibration::chainSamples, Variables::continuous_variables(), Response::copy(), Variables::copy(), NonDQUESOBayesCalibration::copy_gsl_partial(), Model::current_response(), Model::current_variables(), Dakota::data_pairs, Model::evaluate(), Response::function_values(), Model::interface_id(), Dakota::lookup_by_val(), NonDBayesCalibration::mcmcModel, NonDBayesCalibration::mcmcModelHasSurrogate, Model::model_type(), NonD::natafTransform, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, ActiveSet::request_values(), NonDBayesCalibration::standardizedSpace, and ParamResponsePair::variables().

Referenced by NonDQUESOBayesCalibration::run_chain_with_restarting().

**void init_parameter_domain( ) [protected]**

initialize the QUESO parameter space, min, max, initial, and domain

Initialize the calibration parameter domain (paramSpace, paramMins/paramMaxs, paramDomain, param-Initials, priorRV)

References Model::continuous_variables(), NonDQUESOBayesCalibration::copy_gsl_partial(), NonDBayes- Calibration::emulatorType, NonDBayesCalibration::invGammaDists, NonDBayesCalibration::mcmcModel, NonD::natafTransform, NonDQUESOBayesCalibration::nonDQUESOInstance, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, NonDQUESOBayesCalibration::paramDomain, NonDQUESOBayesCalibration::paramInitials, NonDQUESOBayesCalibration::paramSpace, NonDQUESOBayesCalibration::quesoEnv, and NonDBayesCalibration::standardizedSpace.

Referenced by NonDQUESOBayesCalibration::calibrate(), and NonDGPMSABayesCalibration::calibrate().

**void prior_proposal_covariance( ) [protected]**

use covariance of prior distribution for setting proposal covariance

Must be called after paramMins/paramMaxs set above

References NonD::natafTransform, Analyzer::numContinuousVars, Iterator::outputLevel, NonDQUESOBayesCalibration::proposalCovMatrix, and NonDBayesCalibration::standardizedSpace.

**void user_proposal_covariance( const String & input_fmt, const RealVector & cov_data, const String & cov_filename ) [protected]**

set proposal covariance from user-provided diagonal or matrix

This function will convert user-specified cov_type = "diagonal" | "matrix" data from either cov_data or cov_-filename and populate a full QUESO::GslMatrix* in proposalCovMatrix with the covariance.

References Analyzer::numContinuousVars, NonDQUESOBayesCalibration::proposalCovMatrix, and NonDBayesCalibration::standardizedSpace.

**void set_ip_options( ) [protected]**

set inverse problem options callIpOptionsValues common to all solvers

set inverse problem options common to all solvers

References NonDQUESOBayesCalibration::advancedOptionsFile, NonDQUESOBayesCalibration::callIpOptionsValues, Iterator::outputLevel, and NonDQUESOBayesCalibration::quesoEnv.

Referenced by NonDQUESOBayesCalibration::init_queso_solver(), and NonDGPMSABayesCalibration::init_queso_solver().
13.127. **Nondreliability Class Reference**

Base class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDReliability:

```
NonDReliability
  NonD
  NonDAnalyzer
    Iterator
  NonDGlobalReliability
  NonDLocalReliability
```

**Public Member Functions**

- `bool resize ()`
  
  *reinitializes iterator based on new variable size*

**Protected Member Functions**

- `NonDReliability (ProblemDescDB &problem_db, Model &model)`
  
  *constructor*

- `~NonDReliability ()`
  
  *destructor*

- `void initialize_graphics (int iterator_server_id=1)`
  
  *initialize graphics customized for reliability methods*

- `const Model & algorithm_space_model () const`
Protected Attributes

- **Model uSpaceModel**
  
  *Model* representing the limit state in u-space, after any recastings and data fits.

- **Model mppModel**
  
  *RecastModel* which formulates the optimization subproblem: RIA, PMA, EGO.

- **Iterator mppOptimizer**
  
  *Iterator* which optimizes the mppModel.

- **unsigned short mppSearchType**
  
  The MPP search type selection: MV, x/u-space AMV, x/u-space AMV+, x/u-space TANA, x/u-space EGO, or NO A-PPROX

- **Iterator importanceSampler**
  
  Importance sampling instance used to compute/refine probabilities

- **unsigned short integrationRefinement**
  
  Integration refinement type (NO_INT_REFINE, IS, AIS, or MMAIS) provided by refinement specification

- **size_t numRelAnalyses**
  
  Number of invocations of core run()

- **size_t approxIters**
  
  Number of approximation cycles for the current respFnCount/levelCount

- **bool approxConverged**
  
  Indicates convergence of approximation-based iterations

- **int respFnCount**
  
  Counter for which response function is being analyzed

- **size_t levelCount**
  
  Counter for which response/probability level is being analyzed

- **size_t statCount**
  
  Counter for which final statistic is being computed

- **bool pmaMaximizeG**
  
  Flag indicating maximization of \( G(u) \) within PMA formulation

- **Real requestedTargetLevel**
  
  The \{response, reliability, generalized reliability\} level target for the current response function

Additional Inherited Members

13.127.1 Detailed Description

Base class for the reliability methods within DAKOTA/UQ.

The **NonDReliability** class provides a base class for **NonDLocalReliability**, which implements traditional MPP-based reliability methods, and **NonDGlobalReliability**, which implements global limit state search using Gaussian process models in combination with multimodal importance sampling.
13.127.2 Member Function Documentation

const Model & algorithm.space_model( ) const  [inline],[protected],[virtual]
default definition that gets redefined in selected derived Minimizers
Reimplemented from Analyzer.
References NonDReliability::uSpaceModel.
The documentation for this class was generated from the following files:

- NonDReliability.hpp
- NonDReliability.cpp

13.128 NonDRKDDarts Class Reference

Base class for the Recursive k-d Dart methods within DAKOTA/UQ.

Inheritance diagram for NonDRKDDarts:

```
NonDRKDDarts
    |___NonD
        |___Analyzer
            |___Iterator
```

Public Member Functions

- **NonDRKDDarts (ProblemDescDB &problem_db, Model &model)**
  constructor
- **~NonDRKDDarts ()**
  destructor
- **bool resize ()**
  reinitializes iterator based on new variable size
- **void core_run ()**

Protected Member Functions

- **void pre_run ()**
  generate samples
- **void initiate_random_number_generator (unsigned long x)**
- **double generate_a_random_number ()**
- **void init_rkd_darts ()**
- **void create_rkd_containers (size_t expected_num_samples)**
- **void execute ()**
- **void create_initial_children (size_t parent)**
- void create_new_sample (size_t parent, size_t left, size_t right, double position)
- void improve_parent_evaluation (size_t parent)
- void evaluate_1d_surrogate (size_t parent)
- double get_surrogate_interp_error (size_t parent)
- double estimate_surrogate_evaluation_err (size_t parent)
- void get_children (size_t parent, size_t *children)
- void get_neighbors (size_t sample, size_t &num_neighbors, size_t *neighbors)
- double interpolate_lagrange (size_t num_data_points, double *data_x, double *data_f, double x)
- double integrate_legendre_gauss (double xmin, double xmax, size_t num_data_points, double *data_x, double *data_f, double &err_est)
- double f_true (double *x)
- void initialize_surrogates ()
- void compute_response (double *x)
- void add_surrogate_data (const Variables &vars, const Response &resp)
- void build_surrogate ()
- double eval_surrogate (size_t fn_index, double *vin)
- void estimate_rkd_surrogate ()
- void post_run (std::ostream &s)

  - void print_integration_results (std::ostream &s)
  - void exit_rkd_darts ()
  - void destroy_rkd_containers ()

Protected Attributes

- int samples
- int seed
- int emulatorSamples
- double Q[1220]
- int indx
- double cc
- double c
- double zc
- double zx
- double zy
- size_t qlen

Private Attributes

- double *_1_RKD
- bool _eval_error
- size_t _test_function
- size_t _num_inserted_points
- size_t _num_dim
- size_t _num_samples
- size_t _max_num_samples
- size_t _num_evaluations
Additional Inherited Members

13.128.1 Detailed Description

Base class for the Recursive k-d Dart methods within DAKOTA/UQ.

The NonDRKDDart class recursively implements the numerical integration of a domain based on k-d flat samples.

13.128.2 Member Function Documentation

void core_run() [virtual]

Loop over the set of samples and compute responses.
Reimplemented from Iterator.

void pre_run() [protected], [virtual]

generate samples
Generate Parameter Sets.
Reimplemented from NonD.
References NonD::pre_run().

void post_run(std::ostream & s) [protected], [virtual]

generate statistics
Print function evaluation summary, and integration results.
Reimplemented from Analyzer.
References Iterator::iteratedModel, Analyzer::post_run(), and Model::print_evaluation_summary().
The documentation for this class was generated from the following files:

• NonDRKDDarts.hpp
• NonDRKDDarts.cpp
13.129  NonDSampling Class Reference

Base class for common code between NonDLHSSampling, NonDAdaptImpSampling, and other specializations.

Inheritance diagram for NonDSampling:

```
  NonDSampling
  |                  |  NonD
  |                  |  Analyzer
  |                  |  Iterator
  NonDAdaptImpSampling  NonDAdaptiveSampling  NonDGPImpSampling  NonDLHSSampling  NonDMultilevelSampling
```

Public Member Functions

- **NonDSampling (Model &model, const RealMatrix &sample_matrix)**
  alternate constructor for evaluating and computing statistics for the provided set of samples

- **void compute_statistics (const RealMatrix &vars_samples, const IntResponseMap &resp_samples)**
  For the input sample set, computes mean, standard deviation, and probability/reliability/response levels (aleatory uncertainties) or intervals (epistemic or mixed uncertainties)

- **void compute_intervals (RealRealPairArray &extreme_fns)**
  called by compute_statistics() to calculate min/max intervals using allResponses

- **void compute_intervals (const IntResponseMap &samples)**
  called by compute_statistics() to calculate extremeValues from samples

- **void compute_intervals (RealRealPairArray &extreme_fns, const IntResponseMap &samples)**
  called by compute_statistics() to calculate min/max intervals using samples

- **void compute_moments (const RealVectorArray &fn_samples)**
  calculates sample moments from a matrix of observations for a set of QoI

- **void compute_moments (const IntResponseMap &samples)**
  calculate sample moments and confidence intervals from a map of response observations

- **void compute_moments (const IntResponseMap &samples, RealMatrix &moment_stats, RealMatrix &moment_grads, RealMatrix &moment_conf_ints, short moments_type, const StringArray &labels)**
  convert IntResponseMap to RealVectorArray and invoke helpers

- **void compute_moment_gradients (const RealVectorArray &fn_samples, const RealMatrixArray &grad_samples, const RealMatrix &moment_stats, RealMatrix &moment_grads, short moments_type)**
  compute moment grads from function and gradient samples

- **void compute_moment_confidence_intervals (const RealMatrix &moment_stats, RealMatrix &moment_conf_ints, const SizetArray &sample_counts, short moments_type)**
  compute moment confidence intervals from moment values

- **void archive_moments (const RealMatrix &moment_stats, short moments_type, const StringArray &labels)**
  archive moment statistics in results DB

- **void archive_moment_confidence_intervals (const RealMatrix &moment_conf_ints, short moments_type, const StringArray &labels)**
  archive moment confidence intervals in results DB
• void compute_level_mappings (const IntResponseMap &samples)
  called by compute_statistics() to calculate CDF/CCDF mappings of z to p/beta and of p/beta to z as well as PDFs
• void print_statistics (std::ostream &s) const
  prints the statistics computed in compute_statistics()
• void print_intervals (std::ostream &s) const
  prints the intervals computed in compute_intervals() with default qoi_type and moment_labels
• void print_intervals (std::ostream &s, String qoi_type, const StringArray &interval_labels) const
  prints the intervals computed in compute_intervals()
• void print_moments (std::ostream &s) const
  prints the moments computed in compute_moments() with default qoi_type and moment_labels
• void print_moments (std::ostream &s, String qoi_type, const StringArray &moment_labels) const
  prints the moments computed in compute_moments()
• void print_wilks_statistics (std::ostream &s) const
  prints the Wilks statistics
• void update_final_statistics ()
  update_finalStatistics from minValues/maxValues, momentStats, and computedProbLevels/computedRelLevels/computed-RespLevels
• void transform_samples (bool x_to_u=true)
  transform allSamples imported by alternate constructor. This is needed since random variable distribution parameters are not updated until run time and an imported sample matrix is typically in x-space.
• void transform_samples (RealMatrix &sample_matrix, bool x_to_u, int num_samples=0)
  transform the specified samples matrix from x to u or u to x

Static Public Member Functions

• static void compute_moments (const RealVectorArray &fn_samples, SizetArray &sample_counts, RealMatrix &moment_stats, short moments_type, const StringArray &labels)
  core compute_moments() implementation with all data as inputs
• static void compute_moments (const RealVectorArray &fn_samples, RealMatrix &moment_stats, short moments_type)
  alternate RealMatrix samples API for use by external clients
• static void compute_moments (const RealMatrix &fn_samples, RealMatrix &moment_stats, short moments_type)
  core compute_moments() implementation with all data as inputs
• static void compute_moments (const RealMatrix &fn_samples, RealMatrix &moment_stats, short moments_type)
• static void print_moments (std::ostream &s, const RealMatrix &moment_stats, const RealMatrix moment_cis, String qoi_type, short moments_type, const StringArray &moment_labels, bool print_cis)
  core print moments that can be called without object
• static int compute_wilks_sample_size (unsigned short order, Real alpha, Real beta, bool twosided=false)
  calculates the number of samples using the Wilks formula Static so I can test without instantiating a NonDSampling object - RWH
• static Real compute_wilks_residual (unsigned short order, int nsamples, Real alpha, Real beta, bool twosided)
  Helper function - calculates the Wilks residual.
• static Real compute_wilks_alpha (unsigned short order, int nsamples, Real alpha, Real beta, bool twosided=false)
  calculates the alpha parameter given number of samples using the Wilks formula Static so I can test without instantiating a NonDSampling object - RWH
• static Real **compute_wilks_beta** (unsigned short order, int nsamples, Real alpha, bool twosided=false)
  calculates the beta parameter given number of samples using the Wilks formula Static so I can test without instantiating a NonDSampling object - RWH

• static Real **get_wilks_alpha_min** ()
  Get the lower and upper bounds supported by Wilks bisection solves.

• static Real **get_wilks_alpha_max** ()
• static Real **get_wilks_beta_min** ()
• static Real **get_wilks_beta_max** ()

Protected Member Functions

• **NonDSampling** (ProblemDescDB &problem_db, Model &model)
  constructor

• **NonDSampling** (unsigned short method_name, Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern, short sampling_vars_mode)
  alternate constructor for sample generation and evaluation "on the fly"

• **NonDSampling** (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation "on the fly"

• **NonDSampling** (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &means, const RealVector &std_devs, const RealVector &lower_bnds, const RealVector &upper_bnds, RealSymMatrix &correl)
  alternate constructor for sample generation of correlated normals "on the fly"

• ~**NonDSampling** ()
  destructor

• void **pre_run** ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

• void **core_run** ()

• int **num_samples** () const

• void **sampling_reset** (int min_samples, bool all_data_flag, bool stats_flag)
  resets number of samples and sampling flags

• void **sampling_reference** (int samples_ref)
  set reference number of samples, which is a lower bound during reset

• unsigned short **sampling_scheme** () const
  return sampleType

• void **vary_pattern** (bool pattern_flag)
  set varyPattern

• void **get_parameter_sets** (Model &model)
  Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model.

• void **get_parameter_sets** (Model &model, const int num_samples, RealMatrix &design_matrix)
  Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model and populates the specified design matrix.

• void **get_parameter_sets** (Model &model, const int num_samples, RealMatrix &design_matrix, bool write_=msg)
void get parameter sets (const RealVector &lower_bnds, const RealVector &upper_bnds)

Uses lhsDriver to generate a set of uniform samples over lower_bnds/upper_bnds.

void get parameter sets (const RealVector &means, const RealVector &std_devs, const RealVector &lower_bnds, const RealVector &upper_bnds, RealSymMatrix &correl)

Uses lhsDriver to generate a set of normal samples.

void update model from sample (Model &model, const Real *sample_vars)

Override default update of continuous vars only.

void sample to variables (const Real *sample_vars, Variables &vars)

override default mapping of continuous variables only

void variables to sample (const Variables &vars, Real *sample_vars)

const RealVector & response error estimates () const

return error estimates associated with each of the finalStatistics

void initialize lhs (bool write_message, int num_samples)

increments numLHSRuns, sets random seed, and initializes lhsDriver

void active set mapping ()

in the case of sub-iteration, map from finalStatistics.active_set() requests to activeSet used in evaluate parameter sets()

void view design counts (const Model &model, size_t &num_cdv, size_t &num_ddiv, size_t &num_dsv, size_t &num_d DRV) const

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

void view aleatory uncertain counts (const Model &model, size_t &num_auv, size_t &num_dauiv, size_t &num_dausv, size_t &num_dauurv) const

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

void view epistemic uncertain counts (const Model &model, size_t &num_ceuv, size_t &num_deuiv, size_t &num_deusv, size_t &num_deurv) const

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

void view uncertain counts (const Model &model, size_t &num_cuv, size_t &num_duv, size_t &num_dusv, size_t &num_durv) const

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

void view state counts (const Model &model, size_t &num_csv, size_t &num_dsiv, size_t &num_dssv, size_t &num_dsrv) const

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

void mode counts (const Model &model, size_t &cv_start, size_t &num_cv, size_t &div_start, size_t &num_div, size_t &dsv_start, size_t &num_dsv, size_t &drv_start, size_t &num_drv) const

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model
Static Protected Member Functions

- static void accumulate_mean (const RealVectorArray &fn_samples, size_t q, size_t &num_samp, Real &mean)
  
  helper to accumulate sum of finite samples

- static void accumulate_moments (const RealVectorArray &fn_samples, size_t q, short moments_type, Real *moments)
  
  helper to accumulate higher order sums of finite samples

- static void accumulate_moment_gradients (const RealVectorArray &fn_samples, const RealMatrixArray &grad_samples, size_t q, short moments_type, Real mean, Real mom2, Real *mean_grad, Real *mom2_grad)
  
  helper to accumulate gradient sums

Protected Attributes

- const int seedSpec
  the user seed specification (default is 0)

- int randomSeed
  the current seed

- const int samplesSpec
  initial specification of number of samples

- int samplesRef
  reference number of samples updated for refinement

- int numSamples
  the current number of samples to evaluate

- String rngName
  name of the random number generator

- unsigned short sampleType
  the sample type: default, random, lhs, < incremental random, or incremental lhs

- bool wilksFlag
  flags use of Wilks formula to calculate num samples

- unsigned short wilksOrder

- Real wilksAlpha

- Real wilksBeta

- short wilksSidedness

- RealMatrix momentGrads
  gradients of standardized or central moments of response functions, as determined by finalMomentsType. Calculated in compute_moments() and indexed as (var,moment) when moment id runs from 1:2*numFunctions.

- RealVector finalStatErrors
  standard errors (estimator std deviation) for each of the finalStatistics

- int samplesIncrement
  current increment in a sequence of samples

- Pecos::LHSDriver lhsDriver
  the C++ wrapper for the F90 LHS library

- bool statsFlag
flags computation/output of statistics

- **bool allDataFlag**
  flags update of allResponses < (allVariables or allSamples already defined)

- **short samplingVarsMode**
  the sampling mode: ALEATORY, UNCERTAIN[, UNIFORM], EPISTEMIC, UNCERTAIN[, UNIFORM], UNCERTAIN[, UNIFORM], ACTIVE[, UNIFORM], or ALL[, UNIFORM]. This is a secondary control on top of the variables view that allows sampling over subsets of variables that may differ from the view.

- **short sampleRanksMode**
  mode for input/output of LHS sample ranks: IGNORE_RANKS, GET_RANKS, SET_RANKS, or SET_GET_RANKS

- **bool varyPattern**
  flag for generating a sequence of seed values within multiple `get_parameter_sets()` calls so that these executions (e.g., for SBO/SBNLS) are not repeated, but are still repeatable

- **RealMatrix sampleRanks**
  data structure to hold the sample ranks

- **SensAnalysisGlobal nonDSampCorr**
  initialize statistical post processing

- **bool backfillFlag**
  flags whether to use backfill to enforce uniqueness of discrete LHS samples

- **RealRealPairArray extremeValues**
  Minimum and maximum values of response functions for epistemic calculations (calculated in `compute_intervals()`).

### Private Attributes

- **size_t numLHSRuns**
  counter for number of executions of `get_parameter_sets()` for this object

- **RealMatrix momentCIs**
  Matrix of confidence internals on moments, with rows for mean_lower, mean_upper, sd_lower, sd_upper (calculated in `compute_moments()`)

### Additional Inherited Members

#### 13.129.1 Detailed Description

Base class for common code between `NonDLHSSampling`, `NonDAdeptImpSampling`, and other specializations.

This base class provides common code for sampling methods which employ the Latin Hypercube Sampling (LHS) package from Sandia Albuquerque’s Risk and Reliability organization. `NonDSampling` now exclusively utilizes the 1998 Fortran 90 LHS version as documented in SAND98-0210, which was converted to a UNIX link library in

1. The 1970’s vintage LHS (that had been f2e’d and converted to incomplete classes) has been removed.

#### 13.129.2 Constructor & Destructor Documentation

`NonDSampling ( Model & model, const RealMatrix & sample_matrix )`

alternate constructor for evaluating and computing statistics for the provided set of samples

This alternate constructor defines allSamples from an incoming sample matrix.

References `Analyzer::allSamples`, `Analyzer::compactMode`, `Iterator::maxEvalConcurrency`, `NonDSampling::numSamples`, `NonDSampling::samplesRef`, `NonDSampling::samplesSpec`, and `Iterator::subIteratorFlag`.
CHAPTER 13. CLASS DOCUMENTATION

NonDSampling ( ProblemDescDB & problem_db, Model & model ) [protected]

class constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDSampling::compute_wilks_sample_size(), NonD::epistemicStats, ProblemDescDB::get_real(), ProblemDescDB::get_short(), ProblemDescDB::get_ushort(), NonD::initialize_final_statistics(), Iterator::maxEvalConcurrency, Analyzer::numFunctions, NonDSampling::numSamples, Iterator::probDescDB, NonD::requestedProbLevels, NonDSampling::samplesRef, NonDSampling::sampleType, NonD::totalLevelRequests, and NonDSampling::wilksFlag.

NonDSampling ( unsigned short method_name, Model & model, unsigned short sample_type, int samples, int seed, const String & rng, bool vary_pattern, short sampling_vars_mode ) [protected]

alternate constructor for sample generation and evaluation "on the fly"

This alternate constructor is used for generation and evaluation of on-the-fly sample sets.

References NonD::epistemicStats, Iterator::maxEvalConcurrency, NonD::numEpistemicUncVars, NonDSampling::numSamples, NonD::sampleType, NonD::samplingVarsMode, and Iterator::subIteratorFlag.

NonDSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds ) [protected]

alternate constructor for sample generation "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets.

References Iterator::maxEvalConcurrency, NonDSampling::numSamples, NonD::sampleType, and Iterator::subIteratorFlag.

NonDSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & means, const RealVector & std_devs, const RealVector & lower_bnds, const RealVector & upper_bnds, RealSymMatrix & correl ) [protected]

alternate constructor for sample generation of correlated normals "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of normal, correlated sample sets.

References Iterator::maxEvalConcurrency, NonDSampling::numSamples, NonD::sampleType, and Iterator::subIteratorFlag.

13.129.3 Member Function Documentation

void compute_level_mappings ( const IntResponseMap & samples )

called by compute_statistics() to calculate CDF/CCDF mappings of z to p/beta and of p/beta to z as well as PDFs

Computes CDF/CCDF based on sample binning. A PDF is inferred from a CDF/CCDF within compute_densities() after level computation.

References Dakota::abort_handler(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), NonD::archive_allocate_mappings(), NonD::archive_from Resp(), NonD::archive_to Resp(), NonD::cdfFlag, NonD::compute_densities(), NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRelLevels, NonD::computedRespLevels, NonDSampling::extremeValues, NonD::finalMomentsType, NonD::finalStatistics, Response::function_gradient_view(), NonD::initialize_level_mappings(), Iterator::iteratedModel, NonD::momentGrads, NonD::momentStats, Analyzer::numFunctions, NonD::pdfOutput, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and Model::response_labels().
Referenced by NonDExpansion::compute_numerical_statistics(), and NonDSampling::compute_statistics().

```cpp
void transform_samples ( bool xfo_u = true ) [inline]
```
transform allSamples imported by alternate constructor. This is needed since random variable distribution parameters are not updated until run time and an imported sample matrix is typically in x-space.
- transform x_samples to u_samples for use by expansionSampler
- References Analyzer::allSamples, and NonDSampling::numSamples.
- Referenced by NonDLHSSampling::d_optimal_parameter_set(), and NonDExpansion::initialize_expansion().

```cpp
void pre_run ( ) [inline], [protected], [virtual]
```
pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
- Reimplemented from NonD.
- References NonDSampling::active_set_mapping(), NonD::pre_run(), and Iterator::subIteratorFlag.
- Referenced by NonDMultilevelSampling::pre_run(), and NonDLHSSampling::pre_run().

```cpp
void core_run ( ) [protected], [virtual]
```
Default implementation generates allResponses from either allSamples or allVariables.
- Reimplemented from Iterator.
- References NonDSampling::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, and NonDSampling::statsFlag.

```cpp
int num_samples ( ) const [inline], [protected], [virtual]
```
Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.
- Reimplemented from Analyzer.
- References NonDSampling::numSamples.
- Referenced by NonDAdaptImpSampling::evaluate_samples(), NonDSampling::get_parameter_sets(), NonDSampling::initialize_lhs(), NonDSampling::print_wilks_stats(), NonDAdaptImpSampling::select_rep_points(), NonDLHSSampling::store_ranks(), and NonDSampling::transform_samples().

```cpp
void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [inline], [protected], [virtual]
```
resets number of samples and sampling flags
- used by DataFitSurrModel::build_global() to publish the minimum number of samples needed from the sampling routine (to build a particular global approximation) and to set allDataFlag and statsFlag. In this case, allDataFlag is set to true (vectors of variable and response sets must be returned to build the global approximation) and statsFlag is set to false (statistics computations are not needed).
- Reimplemented from Iterator.
- References NonDSampling::allDataFlag, NonDSampling::numSamples, NonDSampling::samplesIncrement, NonDSampling::samplesRef, and NonDSampling::statsFlag.
void get_parameter_sets ( Model & model ) [inline], [protected], [virtual]

Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model.

This version of get_parameter_sets() extracts data from the user-defined model in any of the four sampling modes and populates class member allSamples.

Reimplemented from Analyzer.

References Analyzer::allSamples, and NonDSampling::numSamples.

Referenced by NonDLHSSampling::compute_pca(), NonDAdaptImpSampling::core_run(), NonDLHSSampling::d_optimal_parameter_set(), NonDSampling::get_parameter_sets(), NonDLHSSampling::increment_lhs_parameter_set(), NonDLHSSampling::increment_lhs_set(), NonDMultilevelSampling::increment(), NonDSampling::multilevel_control_variate_mc_Qcorr(), NonDSampling::multilevel_control_variate_mc_Ycorr(), NonDMultilevelSampling::multilevel_mc_Qsum(), NonDMultilevelSampling::multilevel_mc_Ysum(), NonDLHSSampling::multilevel_sampling(), NonDLHSSampling::pre_run(), and NonDMultilevelSampling::shared_increment().

void get_parameter_sets ( Model & model, const int num_samples, RealMatrix & design_matrix ) [inline], [protected], [virtual]

Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model and populates the specified design matrix.

This version of get_parameter_sets() extracts data from the user-defined model in any of the four sampling modes and populates the specified design matrix.

Reimplemented from Analyzer.

References NonDSampling::get_parameter_sets().

void get_parameter_sets ( const RealVector & lower_bnds, const RealVector & upper_bnds ) [protected]

Uses lhsDriver to generate a set of uniform samples over lower_bnds/upper_bnds.

This version of get_parameter_sets() does not extract data from the user-defined model, but instead relies on the incoming bounded region definition. It only support a UNIFORM sampling mode, where the distinction of ACTIVE_UNIFORM vs. ALL_UNIFORM is handled elsewhere.

References Analyzer::allSamples, NonDSampling::initialize_lhs(), NonDSampling::lhsDriver, and NonDSampling::numSamples.

void get_parameter_sets ( const RealVector & means, const RealVector & std_devs, const RealVector & lower_bnds, const RealVector & upper_bnds, RealSymMatrix & correl ) [protected]

Uses lhsDriver to generate a set of normal samples.

This version of get_parameter_sets() does not extract data from the user-defined model, but instead relies on the incoming definition. It only support the sampling of normal variables.

References Analyzer::allSamples, NonDSampling::initialize_lhs(), NonDSampling::lhsDriver, and NonDSampling::numSamples.

void variables_to_sample ( const Variables & vars, Real * sample_vars ) [protected], [virtual]

Map the active variables from vars to sample_vars (column in allSamples)

Reimplemented from Analyzer.

References Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Model::discrete_set_string_values(), Variables::discrete_string_variables(), Iterator::iteratedModel, Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, Dakota::set_value_to_index(), and Variables::view().
void active_set_mapping() [protected]

in the case of sub-iteration, map from finalStatistics.active_set() requests to activeSet used in evaluate_parameter_sets()

Map ASV/DVV requests in final statistics into activeSet for use in evaluate_parameter_sets()

References Response::active_set_derivative_vector(), Response::active_set_request_vector(), Iterator::activeSet, ActiveSet::derivative_vector(), NonD::finalMomentsType, NonD::finalStatistics, Analyzer::numFunctions, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and NonD::totalLevelRequests.

Referenced by NonDSampling::pre_run().

void view_design_counts (const Model & model, size_t & num_cdv, size_t & num_ddiv, size_t & num_ddsv, size_t & num_ddrv) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function computes total design variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References Model::current_variables(), Variables::cv_start(), SharedVariablesData::design_counts(), Variables::div_start(), Variables::drv_start(), Variables::dsv_start(), NonD::numContDesVars, NonD::numDiscIntDesVars, NonD::numDiscRealDesVars, NonD::numDiscStringDesVars, Variables::shared_data(), and Variables::view().

Referenced by NonDSampling::mode_counts().

void view_aleatory_uncertain_counts (const Model & model, size_t & num_cauv, size_t & num_dauiv, size_t & num_dausv, size_t & num_daurv) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function computes total aleatory uncertain variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References SharedVariablesData::aleatory_uncertain_counts(), Model::current_variables(), NonD::numContAleatUncVars, NonD::numDiscIntAleatUncVars, NonD::numDiscRealAleatUncVars, NonD::numDiscStringAleatUncVars, Variables::shared_data(), and Variables::view().

Referenced by NonDSampling::mode_counts().

void view_epistemic_uncertain_counts (const Model & model, size_t & num_ceuv, size_t & num_deuiv, size_t & num_deusv, size_t & num_deurv) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function computes total epistemic uncertain variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References Model::current_variables(), SharedVariablesData::epistemic_uncertain_counts(), NonD::numContEpistUncVars, NonD::numDiscIntEpistUncVars, NonD::numDiscRealEpistUncVars, NonD::numDiscStringEpistUncVars, Variables::shared_data(), and Variables::view().

Referenced by NonDSampling::mode_counts().
CHAPTER 13. CLASS DOCUMENTATION

void view_uncertain_counts ( const Model & model, size_t & num_cv, size_t & num_div, size_t & num_dsv, size_t & num_drv ) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function computes total uncertain variable counts, not active counts, for use in defining offsets and counts within all variables arrays.

References Model::current_variables(), Variables::cv(), Variables::div(), Variables::drv(), NonD::numContAleatUncVars, NonD::numContEpistUncVars, NonD::numDiscIntAleatUncVars, NonD::numDiscIntEpistUncVars, NonD::numDiscRealAleatUncVars, NonD::numDiscRealEpistUncVars, NonD::numDiscStringAleatUncVars, NonD::numDiscStringEpistUncVars, Variables::shared_data(), SharedVariablesData::uncertain_counts(), and Variables::view().

Referenced by NonDSampling::mode_counts().

void mode_counts ( const Model & model, size_t & cv_start, size_t & num_cv, size_t & div_start, size_t & num_div, size_t & dsv_start, size_t & num_dsv, size_t & drv_start, size_t & num_drv ) const [protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function and its helpers to follow are needed since NonDSampling supports a richer set of sampling modes than just the active variable subset. mode_counts() manages the samplingVarsMode setting, while its helper functions (view{design,aleatory_uncertain,epistemic_uncertain, uncertain,state}counts) manage the active variables view. Similar to the computation of starts and counts in creating active variable views, the results of this function are starts and counts for use within model.all_*() set/get functions.

References Model::acv(), Model::adiv(), Model::adriv(), Model::adsv(), Model::current_variables(), Variables::cv(), Variables::cv_start(), Variables::div(), Variables::div_start(), Variables::drv(), Variables::drv_start(), Variables::dsv(), Variables::dsv_start(), NonDSampling::samplingVarsMode, NonDSampling::view_aleatory_uncertain_counts(), NonDSampling::view_design_counts(), NonDSampling::view_epistemic_uncertain_counts(), and NonDSampling::view_uncertain_counts().

Referenced by NonDSampling::compute_statistics(), NonDLHSSampling::d_optimal_parameter_set(), NonDSampling::get_parameter_sets(), NonDLHSSampling::post_input(), NonDLHSSampling::pre_run(), NonDSampling::print_statistics(), NonDSampling::sample_to_variables(), and NonDSampling::update_model_from_sample().

The documentation for this class was generated from the following files:

- NonDSampling.hpp
- NonDSampling.cpp

13.130 NonDSparseGrid Class Reference

Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

Inheritance diagram for NonDSparseGrid:
Public Member Functions

- **NonDSparseGrid** (Model &model, unsigned short ssg_level, const RealVector &dim_pref, short exp_coeffs_soln_approach, short driver_mode, short growth_rate=Pecos::MODERATE_RESTRICTED_GROWTH, short refine_control=Pecos::NO_CONTROL, bool track_uniq_prod_wts=true, bool track_colloc_indices=true)
  - void *sparse_grid_level* (unsigned short ssg_level)
    - update the sparse grid level (e.g., from a level sequence)
  - void *increment_grid* ()
    - increment ssgDriver::ssgLevel
  - void *increment_grid_weights* (const RealVector &aniso_wts)
    - update ssgDriver::ssgAnisoLevelWts and increment ssgDriver::ssgLevel based on specified anisotropic weighting
  - void *reset* ()
    - set level and dimension preference within ssgDriver based on ssgLevelSpec and dimPrefSpec, following refinement or sequence advancement
  - const std::set< UShortArray > & *active_multi_index* () const
    - returns SparseGridDriver::active_multi_index()
  - void *print_smolyak_multi_index* () const
    - invokes SparseGridDriver::print_smolyak_multi_index()
  - void *initialize_sets* ()
    - invokes SparseGridDriver::initialize_sets()
  - void *update_reference* ()
    - invokes SparseGridDriver::update_reference()
  - void *increment_set* (const UShortArray &set)
    - invokes SparseGridDriver::push_trial_set()
  - int *increment_size* () const
    - invokes SparseGridDriver::unique_trial_points()
  - void *restore_set* ()
    - invokes SparseGridDriver::restore_set()
  - void *evaluate_set* ()
    - invokes SparseGridDriver::compute_trial_grid()
  - void *decrement_set* ()
    - invokes SparseGridDriver::pop_trial_set()
CHAPTER 13. CLASS DOCUMENTATION

- void update_sets (const UShortArray &set_star)
  invokes SparseGridDriver::update_sets()
- void finalize_sets (bool output_sets, bool converged_within_tol)
  invokes SparseGridDriver::finalize_sets()
- void evaluate_grid_increment ()
  invokes SparseGridDriver::evaluate_grid_increment()
- int num_samples () const

Protected Member Functions

- NonDSparseGrid (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDSparseGrid ()
  destructor
- void initialize_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)
  initialize integration grid by drawing from polynomial basis settings
- void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)

Private Attributes

- Pecos::SparseGridDriver * ssgDriver
  convenience pointer to the numIntDriver representation
- unsigned short ssgLevelSpec
  the user specification for the Smolyak sparse grid level, rendered anisotropic via dimPrefSpec
- unsigned short ssgLevelRef
  reference point (e.g., lower bound) for the Smolyak sparse grid level maintained within ssgDriver

Additional Inherited Members

13.130.1 Detailed Description

Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

This class is used by NonDPolynomialChaos and NonDStochCollocation, but could also be used for general numerical integration of moments. It employs 1-D Clenshaw-Curtis and Gaussian quadrature rules within Smolyak sparse grids.

13.130.2 Constructor & Destructor Documentation

NonDSparseGrid ( Model & model, unsigned short ssg_level, const RealVector & dim_pref, short exp_coeffs_soln_approach, short driver_mode, short growth_rate = Pecos::MODERATE_RESTRICTED_GROWTH, short refine_control = Pecos::NO_CONTROL, bool track_uniq_prod_wts = true, bool track_colloc_indices = true )

This alternate constructor is used for on-the-fly generation and evaluation of sparse grids within PCE and SC. References NonDIntegration::numIntDriver, and NonDSparseGrid::ssgDriver.
NonDSparseGrid ( ProblemDescDB & problem_db, Model & model ) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not a separate sparse_grid method specification.

References Model::aleatory_distribution_parameters(), NonDIntegration::check_variables(), Iterator::convergenceTol, Model::correction_type(), NonDIntegration::dimPrefSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_short(), ProblemDescDB::get_ushort(), NonDSparseGrid::initialize_grid(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, NonD::natafTransform, NonDIntegration::numIntDriver, Iterator::outputLevel, Iterator::probDescDB, NonDSparseGrid::ssgDriver, and NonDSparseGrid::ssgLevelRef.

13.130.3 Member Function Documentation

int num_samples ( ) const [inline], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References NonDSparseGrid::ssgDriver.

void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the sparse grid routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDSparseGrid::ssgDriver, and NonDSparseGrid::ssgLevelRef.

The documentation for this class was generated from the following files:

- NonDSparseGrid.hpp
- NonDSparseGrid.cpp

13.131 NonDStochCollocation Class Reference

Nonintrusive stochastic collocation approaches to uncertainty quantification.

Inheritance diagram for NonDStochCollocation:
Public Member Functions

- **NonDStochCollocation** (ProblemDescDB &problem_db, Model &model)
  *standard constructor*

- **NonDStochCollocation** (Model &model, short exp_coeffs_approach, unsigned short num_int, const RealVector &dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs)
  *alternate constructor*

- **~NonDStochCollocation** ()
  *destructor*

- **bool resize** ()
  *reinitializes iterator based on new variable size*

Protected Member Functions

- **NonDStochCollocation** (BaseConstructor, ProblemDescDB &problem_db, Model &model)
  *short-cut ctor allowing derived class to replace logic in base class ctor*

- **NonDStochCollocation** (unsigned short method_name, Model &model, short exp_coeffs_approach, bool piecewise_basis, bool use_derivs)
  *short-cut ctor allowing derived class to replace logic in base class ctor*

- **void resolve_inputs** (short &u_space_type, short &data_order)
  *perform error checks and mode overrides*

- **void initialize_u_space_model** ()
  *initialize uSpaceModel polynomial approximations with PCE/SC data*

- **void update_expansion** (size_t index=NPOS)
  *update an expansion; avoids overhead in compute_expansion()*

- **Real compute_covariance_metric** ()
  *compute 2-norm of change in response covariance*

- **Real compute_final_statistics_metric** ()
  *compute 2-norm of change in final statistics*
• void config_integration (unsigned short quad_order, unsigned short ssg_level, const RealVector &dim_pref, short u_space_type, Iterator &u_space_sampler, Model &g_u_model)
  configure u_space_sampler based on numerical integration specification
• void config_integration (short exp_coeffs_approach, unsigned short num_int, const RealVector &dim_pref, Iterator &u_space_sampler, Model &g_u_model)
  configure u_space_sampler based on expansion coefficients approach
• void config_approximation_type (String &approx_type)
  define approx_type based on expansion settings

Additional Inherited Members

13.131.1 Detailed Description

Nonintrusive stochastic collocation approaches to uncertainty quantification.

The NonDStochCollocation class uses a stochastic collocation (SC) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the InterpPolyApproximation class to manage multidimensional Lagrange polynomial interpolants.

13.131.2 Constructor & Destructor Documentation

NonDStochCollocation ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Response::active_set(), Model::assign_rep(), ParallelLibrary::command_line_check(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), NonDExpansion::construct_expansion_sampler(), Model::current_response(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_string(), ProblemDescDB::get_realVector(), NonDExpansion::initialize_random(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, Iterator::probDescDB, ActiveSet::request_values(), NonDStochCollocation::resolve_inputs(), NonD::transform_model(), and NonDExpansion::uSpaceModel.

NonDStochCollocation ( Model & model, short exp_coeffs_approach, unsigned short num_int, const RealVector & dim_pref, short u_space_type, bool piecewise_basis, bool use_derivs )

alternate constructor

This constructor is used for helper iterator instantiation on the fly.

References Response::active_set(), Model::assign_rep(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), Model::current_response(), NonDExpansion::initialize_random(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, ActiveSet::request_values(), NonDStochCollocation::resolve_inputs(), NonD::transform_model(), and NonDExpansion::uSpaceModel.

NonDStochCollocation ( BaseConstructor , ProblemDescDB & problem_db, Model & model )

[protected]

short-cut ctor allowing derived class to replace logic in base class ctor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.
NonDStochCollocation (unsigned short method_name, Model & model, short exp_coeffs_approach, bool piecewise_basis, bool use_derivs) [protected]

short-cut ctor allowing derived class to replace logic in base class ctor
This constructor is used for helper iterator instantiation on the fly.

13.131.3 Member Function Documentation

Real compute_covariance_metric() [protected], [virtual]
compute 2-norm of change in response covariance
computes the default refinement metric based on change in respCovariance
Reimplemented from NonDExpansion.
References Model::approximations(), NonDExpansion::compute_covariance_metric(), PecosApproximation::delta_covariance(), PecosApproximation::expansion_coefficient_flag(), NonDExpansion::expansionBasisType, NonDExpansion::initialPtU, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Analyzer::numFunctions, NonDExpansion::respCovariance, and NonDExpansion::uSpaceModel.

Real compute_final_statistics_metric() [protected], [virtual]
compute 2-norm of change in final statistics
computes a "goal-oriented" refinement metric employing finalStatistics
Reimplemented from NonDExpansion.
References Model::approximations(), NonD::cdfFlag, NonDExpansion::compute_final_statistics_metric(), NonDExpansion::compute_statistics(), PecosApproximation::delta_beta(), PecosApproximation::delta_z(), PecosApproximation::expansion_coefficient_flag(), NonDExpansion::expansionBasisType, NonD::finalStatistics, Response::function_values(), NonDExpansion::initialPtU, Response::num_functions(), NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and NonDExpansion::uSpaceModel.

The documentation for this class was generated from the following files:

- NonDStochCollocation.hpp
- NonDStochCollocation.cpp

13.132 NonDWasabiBayesCalibration Class Reference

WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference.
Inheritance diagram for NonDWasabiBayesCalibration:
Public Member Functions

- **NonDWASABIBayesCalibration** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- ~**NonDWASABIBayesCalibration** ()
  
  *destructor*

- void **compute_responses** (RealMatrix &samples, RealMatrix &responses)

Static Public Member Functions

- static void **problem_size** (int &chain_num, int &cr_num, int &gen_num, int &pair_num, int &par_num)
  
  *initializer for problem size characteristics in WASABI*

- static void **problem_value** (std::string *chain_filename, std::string *gr_filename, double &gr_threshold, int &jumpstep, double limits[], int par_num, int &printstep, std::string *restart_read_filename, std::string *restart_write_filename)
  
  *Filename and data initializer for WASABI.*

Protected Member Functions

- void **calibrate** ()

- void **print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  
  *print the final iterator results*

- void **extract_selected_posterior_samples** (const std::vector<int> &points_to_keep, const RealMatrix &samples_for_posterior_eval, const RealVector &posterior_density, RealMatrix &posterior_data) const
  
  *Extract a subset of samples for posterior eval according to the indices in points_to_keep.*

- void **export_posterior_samples_to_file** (const std::string filename, const RealMatrix &posterior_data) const
  
  *Export posterior data to file.*
Protected Attributes

- int numPushforwardSamples
  number of samples from the prior that is pushed forward through the model
- RealVector dataDistMeans
  The mean of the multivariate Gaussian distribution of the obs. data.
- RealVector dataDistCovariance
  The covariance of the multivariate Gaussian distribution of the obs. data.
- std::string dataDistFilename
  The filename of the file containing the data that with density estimator defines the distribution of the obs. data.
- std::string dataDistCovType
  The type of covariance data provided ("diagonal","matrix")
- std::string posteriorSamplesImportFile
  The filename of the import file containing samples at which the posterior will be evaluated.
- unsigned short posteriorSamplesImportFormat
  Format of imported posterior samples file.
- std::string exportPosteriorDensityFile
  The filename of the export file containing an arbitrary set of samples and their corresponding density values.
- std::string exportPosteriorSamplesFile
  The filename of the export file containing samples from the posterior and their corresponding density values.
- unsigned short exportFileFormat
  Format of imported posterior samples and values file.
- bool generateRandomPosteriorSamples
  Flag specifying whether to generate random samples from the posterior.
- bool evaluatePosteriorDensity
  Flag specifying whether to evaluate the posterior density at a set of samples.
- RealVector paramMins
  lower bounds on calibrated parameters
- RealVector paramMaxs
  upper bounds on calibrated parameters
- boost::mt19937 rnumGenerator
  random number engine for sampling the prior
- RealMatrix momentStatistics
  Matrix for moment statistics. Note that posterior values have density associated with them so we can’t use the compute moments in NonDSampling.

Additional Inherited Members

13.132.1 Detailed Description

WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference.
This class performs Bayesian calibration using the WASABI approach.
13.132.2 Constructor & Destructor Documentation

NonDWASABIBayesCalibration ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

13.132.3 Member Function Documentation

void calibrate ( ) [protected], [virtual]

Perform the uncertainty quantification

Implements NonDBayesCalibration.

References Dakota::abort_handler(), Dakota::copy_data(), NonDWASABIBayesCalibration::dataDistCovariance, NonDWASABIBayesCalibration::dataDistFilename, NonDWASABIBayesCalibration::dataDistMeans, NonDBayesCalibration::emulatorType, NonDWASABIBayesCalibration::evaluatePosteriorDensity, NonDWASABIBayesCalibration::export_posterior_samples_to_file(), NonDWASABIBayesCalibration::exportPosteriorDensityFile, NonDWASABIBayesCalibration::exportPosteriorSamplesFile, NonDWASABIBayesCalibration::extract_selected_posterior_samples(), NonD::generate_system_seed(), NonDWASABIBayesCalibration::generateRandomPosteriorSamples, NonDBayesCalibration::initialize_model(), NonDWASABIBayesCalibration::momentStatistics, NonD::natafTransform, Analyzer::numContinuousVars, Analyzer::numFunctions, NonDWASABIBayesCalibration::numPushforwardSamples, Iterator::outputLevel, NonDWASABIBayesCalibration::paramMaxs, NonDWASABIBayesCalibration::paramMins, NonDWASABIBayesCalibration::posteriorSamplesImportFile, NonDBayesCalibration::prior_density(), NonDBayesCalibration::prior_sample(), NonDWASABIBayesCalibration::randomSeed, NonDWASABIBayesCalibration::numGenerator, and NonDBayesCalibration::standardizedSpace.

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from NonDBayesCalibration.

References Model::current_response(), Response::function_labels(), NonDBayesCalibration::mcmcModel, NonDWASABIBayesCalibration::momentStatistics, and NonDSampling::print_moments().

The documentation for this class was generated from the following files:

- NonDWASABIBayesCalibration.hpp
- NonDWASABIBayesCalibration.cpp

13.133 NonlinearCGOptimizer Class Reference

Inheritance diagram for NonlinearCGOptimizer:
Public Member Functions

- **NonlinearCGOptimizer (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **~NonlinearCGOptimizer ()**
  
  *destructor*

- **Real linesearch_eval (const Real &trial_step, short req_val=1)**
  
  *evaluate the objective function given a particular step size (public for use in boost::ls::eval functor; could use friend)*

Protected Member Functions

- **void core_run ()**
  
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

Private Member Functions

- **void parse_options ()**
  
  *constructor helper function to parse misc_options from ProblemDescDB*

- **void compute_direction ()**
  
  *compute next direction via choice of method*

- **bool compute_step ()**
  
  *compute step: fixed, simple decrease, sufficient decrease*

- **void bracket_min (Real &xa, Real &xb, Real &xc, Real &fa, Real &fb, Real &fc)**
  
  *bracket the 1-D minimum in the linesearch*

- **Real brent_minimize (Real a, Real b, Real tol)**
  
  *Perform 1-D minimization for the stepLength using Brent’s method.*

Private Attributes

- **Real initialStep**
  
  *initial step length*

- **Real linesearchTolerance**
  
  *approximate accuracy of absissa in LS*

- **unsigned linesearchType**
  
  *type of line search (if any)*
• unsigned maxLinesearchIters
  maximum evaluations in line search
• Real relFunctionTol
  stopping criterion for rel change in fn
• Real relGradientTol
  stopping criterion for rel reduction in g
• bool resetStep
  whether to reset step with each linesearch
• unsigned restartIter
  iter at which to reset to steepest descent
• unsigned updateType
  type of CG direction update
• unsigned iterCurr
  current iteration number
• RealVector designVars
  current decision variables in the major iteration
• RealVector trialVars
  decision variables in the linesearch
• Real functionCurr
  current function value
• Real functionPrev
  previous function value
• RealVector gradCurr
  current gradient
• RealVector gradPrev
  previous gradient
• RealVector gradDiff
  temporary for gradient difference (gradCurr - gradPrev)
• RealVector searchDirection
  current aggregate search direction
• Real stepLength
  current step length parameter alpha
• Real gradDotGrad_init
  initial gradient norm squared
• Real gradDotGrad_curr
  gradCurr dot gradCurr
• Real gradDotGrad_prev
  gradPrev dot gradPrev

Additional Inherited Members

13.133.1 Detailed Description

Experimental implementation of nonlinear CG optimization
13.133.2 Member Function Documentation

```cpp
void core_run ( ) [protected], [virtual]
```

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.

References Iterator::activeSet, Iterator::bestResponseArray, NonlinearCGOptimizer::compute_direction(), NonlinearCGOptimizer::compute_step(), Model::continuous_variables(), Iterator::convergence-Tol, Dakota::copy_data(), Model::current_response(), NonlinearCGOptimizer::designVars, Model::evaluate(), Response::function_gradient_copy(), Response::function_gradient_view(), Response::function_values(), NonlinearCGOptimizer::functionCurr, NonlinearCGOptimizer::functionPrev, NonlinearCGOptimizer::gradCurr, NonlinearCGOptimizer::gradDotGrad_curr, NonlinearCGOptimizer::gradDotGrad_init, NonlinearCGOptimizer::gradDotGrad_prev, NonlinearCGOptimizer::gradPrev, Iterator::iteratedModel, NonlinearCGOptimizer::iterCurr, NonlinearCGOptimizer::linesearch- Type, Optimizer::localObjectiveRecast, Iterator::maxIterations, Minimizer::numContinuousVars, Iterator::outputLevel, NonlinearCGOptimizer::relFunctionTol, NonlinearCGOptimizer::relGradientTol, ActiveSet::request_values(), NonlinearCGOptimizer::searchDirection, NonlinearCGOptimizer::stepLength, and NonlinearCGOptimizer::trial-Vars.

Real brent_minimize ( Real a, Real b, Real tol ) [private]

Perform 1-D minimization for the stepLength using Brent’s method.
Perform 1-D minimization for the stepLength using Brent’s method. This is a C translation of fmin.f from Netlib.
References NonlinearCGOptimizer::linesearch_eval(), NonlinearCGOptimizer::maxLinesearchIters, and Iterator::outputLevel.
Referenced by NonlinearCGOptimizer::compute_step.
The documentation for this class was generated from the following files:

- NonlinearCGOptimizer.hpp
- NonlinearCGOptimizer.cpp

13.134 NonlinearCGTraits Class Reference

A version of TraitsBase specialized for NonlinearCG optimizers.
Inheritance diagram for NonlinearCGTraits:

```
  TraitsBase
     |  
     v  
NonlinearCGTraits
```

Public Member Functions

- **NonlinearCGTraits ()**
  
  *default constructor*

- **virtual ~NonlinearCGTraits ()**
  
  *destructor*
13.135. NOWPACBLACKBOXEVALUATOR CLASS REFERENCE

- virtual bool is-derived ()
  A temporary query used in the refactor.
- bool supports_continuous_variables ()
  Return the flag indicating whether method supports continuous variables.

13.134.1 Detailed Description
A version of TraitsBase specialized for NonlinearCG optimizers.
The documentation for this class was generated from the following file:
- NonlinearCGOptimizer.hpp

13.135 NOWPACBlackBoxEvaluator Class Reference
Derived class for plugging Dakota evaluations into NOWPAC solver.
Inherits BlackBoxBaseClass.

Public Member Functions
- NOWPACBlackBoxEvaluator (Model &model) constructor
- void evaluate (std::vector<double> const &x, std::vector<double> &vals, void *param)
- void evaluate (std::vector<double> const &x, std::vector<double> &vals, std::vector<double> &noise, void *param)
- void allocate_constraints ()
- int num_ineq_constraints () const
- const SizetList & nonlinear_inequality_mapping_indices () const
- const RealList & nonlinear_inequality_mapping_multipliers () const
- const RealList & nonlinear_inequality_mapping_offsets () const
- void set_unscaled_bounds (const RealVector &l_bnds, const RealVector &u_bnds)
  set \{lower,upper\} Bounds
- void scale (const RealVector &unscaled_x, RealArray &scaled_x) const
  perform scaling from \{lower,upper\} to \{0,1\}
- void unscale (const RealArray &scaled_x, RealVector &unscaled_x) const
  invert scaling to return from \{0,1\} to \{lower,upper\}

Private Attributes
- Model iteratedModel
  cache a local copy of the Model
- RealVector lowerBounds
  cache the active continuous lower bounds for scaling to \{0,1\}
- RealVector upperBounds
  cache the active continuous upper bounds for scaling to \{0,1\}
- int numNowpacIneqConstr
  aggregate unsupported constraint types as nonlinear inequalities
• SizetList nonlinIneqConMappingIndices
  a list of indices for referencing the DAKOTA nonlinear inequality constraints used in computing the corresponding NOWPAC constraints.
• RealList nonlinIneqConMappingMultipliers
  a list of multipliers for mapping the DAKOTA nonlinear inequality constraints to the corresponding NOWPAC constraints.
• RealList nonlinIneqConMappingOffsets
  a list of offsets for mapping the DAKOTA nonlinear inequality constraints to the corresponding NOWPAC constraints.
• SizetList linIneqConMappingIndices
  a list of indices for referencing the DAKOTA linear inequality constraints used in computing the corresponding NOWPAC constraints.
• RealList linIneqConMappingMultipliers
  a list of multipliers for mapping the DAKOTA linear inequality constraints to the corresponding NOWPAC constraints.
• RealList linIneqConMappingOffsets
  a list of offsets for mapping the DAKOTA linear inequality constraints to the corresponding NOWPAC constraints.

13.135.1 Detailed Description
Derived class for plugging Dakota evaluations into NOWPAC solver.
The documentation for this class was generated from the following files:
• NOWPACOptimizer.hpp
• NOWPACOptimizer.cpp

13.136 NOWPACOptimizer Class Reference
Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT)
Inheritance diagram for NOWPACOptimizer:

```
        Iterator
         |   
         v   
Minimizer
        |   
        v   
Optimizer
        |   
        v   
NOWPACOptimizer
```

Public Member Functions
• NOWPACOptimizer (ProblemDescDB &problem_db, Model &model)
  standard constructor
• NOWPACOptimizer (Model &model)
alternate constructor
• \~NOWPACOptimizer ()

destructor
• void core_run ()
core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Private Member Functions
• void initialize_options ()

Shared constructor code.

Private Attributes
• NOWPAC nowpacSolver
• NOWPACBlackBoxEvaluator nowpacEvaluator

Additional Inherited Members

Detailed Description
Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT)

Member Function Documentation
void core_run ( ) [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.

References Iterator::bestResponseArray, Iterator::bestVariablesArray, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Iterator::iteratedModel, Optimizer::localObjectiveRecast, Minimizer::numContinuousVars, Minimizer::numFunctions, Iterator::outputLevel, Model::primary_response_fn_sense(), NOWPACBlackBoxEvaluator::scale(), NOWPACBlackBoxEvaluator::set_unscaled_bounds(), and NOWPACBlackBoxEvaluator::unscale().
The documentation for this class was generated from the following files:
• NOWPACOptimizer.hpp
• NOWPACOptimizer.cpp

NOWPACTraits Class Reference

A version of TraitsBase specialized for NOWPAC optimizers.

Inheritance diagram for NOWPACTraits:

```
+-----------------------------+
| TraitsBase                 |
+-----------------------------+
| \_ NOWPACTraits            |
```
Public Member Functions

- **NOWPACTraits ()**
  - *default constructor*
- virtual **~NOWPACTraits ()**
  - *destructor*
- virtual bool **is derived ()**
  - *A temporary query used in the refactor.*
- bool **supports_continuous_variables ()**
  - *Return the flag indicating whether method supports continuous variables.*
- bool **supports_linear_inequality ()**
  - *Return the flag indicating whether method supports linear inequalities.*
- bool **supports_nonlinear_inequality ()**
  - *Return the flag indicating whether method supports nonlinear inequalities.*

13.137.1 Detailed Description

A version of TraitsBase specialized for NOWPAC optimizers.

The documentation for this class was generated from the following file:

- NOWPACOptimizer.hpp

13.138 NPSOLTraits Class Reference

Wrapper class for the NPSOL optimization library.

Inheritance diagram for NPSOLTraits:

```
+ TraitsBase
|   |
+ NPSOLTraits
```

Public Member Functions

- **NPSOLTraits ()**
  - *default constructor*
- virtual **~NPSOLTraits ()**
  - *destructor*
- virtual bool **is derived ()**
  - *A temporary query used in the refactor.*
- bool **supports_continuous_variables ()**
  - *Return the flag indicating whether method supports continuous variables.*
- bool **supports_linear_inequality ()**
  - *Return the flag indicating whether method supports linear equalities.*
- bool **supports_linear_inequality ()**
  - *Return the flag indicating whether method supports linear inequalities.*
Return the flag indicating whether method supports linear inequalities.

- bool supports_nonlinear_equality ()

Return the flag indicating whether method supports nonlinear equalities.

- bool supports_nonlinear_inequality ()

Return the flag indicating whether method supports nonlinear inequalities.

- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()

Return the format used for nonlinear inequality constraints.

13.138.1 Detailed Description

Wrapper class for the NPSOL optimization library.

The NPSOLOptimizer class provides a wrapper for NPSOL, a Fortran 77 sequential quadratic programming library from Stanford University marketed by Stanford Business Associates. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows: max_function_evaluations is implemented directly in NPSOLOptimizer’s evaluator functions since there is no NPSOL parameter equivalent, and max_iterations, convergence_tolerance, output verbosity, verify_level, function_precision, and linesearch_tolerance are mapped into NPSOL’s “Major Iteration Limit”, “Optimality Tolerance”, “Major Print Level” (verbose: Major Print Level = 20; quiet: Major Print Level = 10), “Verify Level”, “Function Precision”, and “Linesearch Tolerance” parameters, respectively, using NPSOL’s npoptn() subroutine (as wrapped by npoptn2() from the npoptn wrapper.f file). Refer to [Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., 1986] for information on NPSOL’s optional input parameters and the npoptn() subroutine. A version of TraitsBase specialized for NPSOL optimizers.

The documentation for this class was generated from the following file:

- NPSOLOptimizer.hpp

13.139 OptDartsOptimizer Class Reference

Wrapper class for OptDarts Optimizer.

Inheritance diagram for OptDartsOptimizer:

```
  Optimizer
    |__ Minimizer
        |__ Iterator
            |__ OptDartsOptimizer
```

Public Member Functions

- OptDartsOptimizer (ProblemDescDB &problem_db, Model &model)
  Constructor.
• **OptDartsOptimizer** (Model &model)
  
  alternate constructor for *Iterator* instantiations by name

• ∼OptDartsOptimizer ()
  
  Destructor.

• void core_run ()
  
  Calls the OptDarts algorithm.

**Private Member Functions**

• void load_parameters (Model &model)
  
  Convenience function for Parameter loading.

• double opt_darts_f ()
  
  Function evaluation.

• void opt_darts_execute (size_t num_dim, size_t budget, double *xmin, double *xmax, double TOL, size_t problem_index, double fw_MC, double fb_MC)
  
  Run the OPT-DARTS method.

• void opt_darts_initiate (double *xmin, double *xmax)
  
  Initialize OPT-DARTS.

• void opt_darts_reset_convex_hull ()

• size_t opt_darts_pick_candidate (size_t ifunc)
  
  Choose the next trial iterate.

• void retrieve_extended_neighbors (size_t icandidate)

• void opt_darts_sample_from_candidate_neighborhood (size_t icandidate, size_t ifunc)

• void DIRECT_sample_from_candidate_neighborhood (size_t icandidate)

• void opt_darts_add_dart ()

• void opt_darts_update_K_h_approximate_Voronoi (size_t isample)

• void opt_darts_terminate ()
  
  Release memory and exit cleanly.

• void opt_darts_plot_discs_2d (size_t icandidate)
  
  Convenience function for plotting iterates.

• void opt_darts_plot_hull_2d (size_t icandidate, size_t ifunc)
  
  Convenience function for plotting convex hull.

• void initiate_random_generator (unsigned long x)

• double generate_a_random_number ()

• void sample_uniformly_from_unit_sphere_surface (double *dart, size_t num_dim)

• bool trim_line_using_Hyperplane (size_t num_dim, double *st, double *end, double *qH, double *nH)

**Private Attributes**

• double * _xmin

• double * _xmax

• double * _dart

• double * _st

• double * _end

• double * _tmp_point
13.139. OPTDARTSOPTIMIZER CLASS REFERENCE

- double * _qH
- double * _nH
- double ** _x
- double ** _xc
- double ** _f
- double ** _K
- double * _h
- double * _r
- size_t ** _neighbors
- size_t * _tmp_neighbors
- size_t * _ext_neighbors
- size_t _num_ext_neighbors
- bool _use_opt_darts
- bool _estimate_K
- size_t _ib
- size_t _num_samples
- size_t _budget
- size_t _num_dim
- double _diag
- size_t _problem_index
- double _fb
- double _fw
- double _fval
- size_t _corner_index
- size_t _num_corners
- size_t * _corners
- double _epsilon
- double _fb_MC
- double _fw_MC
- double ** _xm
- double ** _xp
- double * _alpha_Deceptive
- double Q [1220]
- int index
- double cc
- double c
- double zc
- double zx
- double zy
- size_t qlen
- bool _use_DIRECT
- int numTotalVars
- int randomSeed
- int maxBlackBoxEvals
- int maxIterations
Additional Inherited Members

13.139.1 Detailed Description

Wrapper class for OptDarts Optimizer.

The documentation for this class was generated from the following files:

- OptDartsOptimizer.hpp
- OptDartsOptimizer.cpp

13.140 OptDartsTraits Class Reference

A version of TraitsBase specialized for OptDarts.

Inheritance diagram for OptDartsTraits:

```
TraitsBase
    ▼
   /    
OptDartsTraits
```

Public Member Functions

- OptDartsTraits ()
  *default constructor*

- virtual ~OptDartsTraits ()
  *destructor*

- virtual bool is_derived ()
  
  *A temporary query used in the refactor.*

- bool supports_continuous_variables ()
  
  *Return the flag indicating whether method supports continuous variables.*

13.140.1 Detailed Description

A version of TraitsBase specialized for OptDarts.

The documentation for this class was generated from the following file:

- OptDartsOptimizer.hpp

13.141 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy.

Inheritance diagram for Optimizer:
Public Member Functions

- `int num_nonlin_ineq_constraints_found () const`
- `template<typename AdapterT> bool get_variable_bounds_from_dakota (typename AdapterT::VecT &lower, typename AdapterT::VecT &upper)`
- `template<typename VecT> void get_responses_from_dakota (const RealVector &dak_fn_vals, VecT &funs, VecT &cEqs, VecT &cIneqs)`

Static Public Member Functions

- `static void not_available (const std::string &package_name)`
  
  *Static helper function: third-party opt packages which are not available.*

Protected Member Functions

- `Optimizer (std::shared_ptr< TraitsBase > traits)`
  
  *default constructor*
- `Optimizer (ProblemDescDB &problem_db, Model &model, std::shared_ptr< TraitsBase > traits)`
  
  *alternate constructor; accepts a model*
- `Optimizer (unsigned short method_name, Model &model, std::shared_ptr< TraitsBase > traits)`
  
  *alternate constructor for "on the fly" instantiations*
- `Optimizer (unsigned short method_name, size_t num_cv, size_t num_div, size_t num_dsv, size_t num_driv, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq, std::shared_ptr< TraitsBase > traits)`
  
  *alternate constructor for "on the fly" instantiations*
- `~Optimizer ()`
  
  *destructor*
- `void initialize_run ()`
- `void post_run (std::ostream &s)`
- `void finalize_run ()`
  
  *utility function to perform common operations following post_run(); deallocation and resetting of instance pointers*
- `void print_results (std::ostream &s, short results_state=FINAL_RESULTS)`
- `void configure_constraint_maps ()`
- `int configure_inequality_constraints (CONSTRAINT_TYPE ctype)`
• void configure_equality_constraints (CONSTRAINT_TYPE ctype, size_t index_offset)
• template<typename AdapterT>
  void get_linear_constraints_and_bounds (typename AdapterT::VecT &lin_ineq_lower_bnds, typename AdapterT::VecT &lin_ineq_upper_bnds, typename AdapterT::VecT &lin_eq_targets, typename AdapterT::MatT &lin_ineq_coeffs, typename AdapterT::MatT &lin_eq_coeffs)

Protected Attributes

• size_t numObjectiveFns
  number of objective functions (iterator view)
• bool localObjectiveRecast
  flag indicating whether local recasting to a single objective is used
• Optimizer * prevOptInstance
  pointer containing previous value of optInstance
• int numNonlinearIneqConstraintsFound
  number of nonlinear ineq constraints actually used (based on conditional and bigRealBoundSize
• std::vector<int> constraintMapIndices
  map from Dakota constraint number to APPS constraint number
• std::vector<double> constraintMapMultipliers
  multipliers for constraint transformations
• std::vector<double> constraintMapOffsets
  offsets for constraint transformations

Static Protected Attributes

• static Optimizer * optimizerInstance
  pointer to Optimizer instance used in static member functions

Private Member Functions

• void reduce_model (bool local_nls_recast, bool require_hessians)
  Wrap iteratedModel in a RecastModel that performs (weighted) multi-objective or sum-of-squared residuals transformation.
• void objective_reduction (const Response &full_response, const BoolDeque &sense, const RealVector &full_wts, Response &reduced_response) const
  forward mapping: maps multiple primary response functions to a single weighted objective for single-objective optimizers

Static Private Member Functions

• static void primary_resp.reducer (const Variables &full_vars, const Variables &reduced_vars, const Response &full_response, Response &reduced_response)
  Recast callback to reduce multiple objectives or residuals to a single objective, with gradients and Hessians as needed.
Additional Inherited Members

13.141.1 Detailed Description

Base class for the optimizer branch of the iterator hierarchy.

The Optimizer class provides common data and functionality for DOTOptimizer, CONMINOptimizer, NPSOLOptimizer, SNLLOptimizer, NLPLOptimizer, COLINOptimizer, OptDartsOptimizer, NCSUOptimizer, NonlinearCGOptimizer, NomadOptimizer, and JEGAOptimizer.

13.141.2 Member Function Documentation

bool get_variable_bounds_from_dakota ( typename AdapterT::VecT & lower, typename AdapterT::VecT & upper ) [inline]

Adapter for transferring variable bounds from Dakota data to TPL data
References Minimizer::bigIntBoundSize, Minimizer::bigRealBoundSize, and Iterator::iteratedModel.

void get_responses_from_dakota ( const RealVector & dak_fn_vals, VecT & funs, VecT & cEqs, VecT & cIneqs ) [inline]

Adapter for transferring responses from Dakota data to TPL data
References Optimizer::constraintMapIndices, Optimizer::constraintMapMultipliers, Optimizer::constraintMapOffsets, Dakota::get_responses(), and Iterator::iteratedModel.
Referenced by APPSEvalMgr::recv().

void initialize_run ( ) [protected], [virtual]

Implements portions of initialize_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of initialize_run() (which would otherwise hide it).
Reimplemented from Minimizer.
References Optimizer::configure_constraint_maps(), Minimizer::initialize_run(), Model::is_null(), Iterator::iteratedModel, Iterator::myModelLayers, Optimizer::optimizerInstance, Optimizer::prevOptInstance, and Model::update_from_subordinate_model().

void post_run ( std::ostream & s ) [protected], [virtual]

Implements portions of post_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of post_run() (which would otherwise hide it).
Reimplemented from Minimizer.
References Dakota::abort_handler(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Variables::continuous_variables(), ScalingModel::cv_scaled2native(), Minimizer::local_recast_retrieve(), Optimizer::localObjectiveRecast, Model::model_rep(), Minimizer::post_run(), ScalingModel::resp_scaled2native(), Minimizer::scaleFlag, and Minimizer::scalingModel.

void finalize_run ( ) [inline], [protected], [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.
Reimplemented from Minimizer.
References Minimizer::finalize_run(), Optimizer::optimizerInstance, and Optimizer::prevOptInstance.
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
Redefines default iterator results printing to include optimization results (objective functions and constraints). Reimplemented from Iterator.

References Dakota::abort_handler(), Minimizer::archive_allocate_best, Minimizer::archive_best, Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, Dakota::data_pairs, Minimizer::dataTransformModel, Model::interface_id, Dakota::lookup_by_val, Model::modelRep, Model::num_functions, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, Minimizer::optimizationFlag, Minimizer::original_model, Model::primary_response_fn_weights, DataTransformModel::print_best_responses, and Minimizer::print_residuals().

void configure_constraint_maps ( ) [protected]
Implements configuration of constraint maps, etc...

References Dakota::abort_handler(), Minimizer::bigRealBoundSize, Dakota::configure_inequality_constraint_maps, Optimizer::constraintMapIndices, Optimizer::constraintMapMultipliers, Optimizer::constraintMapOffsets, Iterator::iteratedModel, Optimizer::numNonlinearIneqConstraintsFound, and Iterator::traits().

Referenced by Optimizer::initialize_run().

void reduce_model ( bool local_nls_recast, bool require_hessians ) [private]
Wrap iteratedModel in a RecastModel that performs (weighted) multi-objective or sum-of-squared residuals transformation.

Reduce model for least-squares or multi-objective transformation. Doesn’t map variables, or secondary responses. Maps active set for Gauss-Newton. Maps primary responses to single objective so user vs. iterated matters.

References Iterator::activeSet, Model::assign_rep, Minimizer::calibrationDataFlag, Model::current_response, Response::function_gradients, Iterator::gnewton_set_recast, Model::hessian_type, Iterator::iteratedModel, Iterator::myModelLayers, Minimizer::numContinuousVars, Minimizer::numFunctions, Minimizer::numIterPrimaryFns, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Optimizer::numObjectiveFns, Minimizer::numTotalCalibTerms, Minimizer::numUserPrimaryFns, Iterator::outputLevel, Model::primary_fn_type, Optimizer::primary_resp_reducer, Model::primary_response_fn_sense, Model::primary_response_fn_weights, ActiveSet::request_vector, and Response::reshape().

Referenced by Optimizer::optimizer().

void primary_resp_reducer ( const Variables & full_vars, const Variables & reduced_vars, const Response & full_response, Response & reduced_response ) [static], [private]
Recast callback to reduce multiple objectives or residuals to a single objective, with gradients and Hessians as needed.

Objective function map from multiple primary responses (objective or residuals) to a single objective. Currently supports weighted sum; may later want more general transformations, e.g., goal-oriented.

References Iterator::iteratedModel, Optimizer::objective_reduction, Optimizer::optimizerInstance, Iterator::outputLevel, Model::primary_response_fn_sense, Model::primary_response_fn_weights, and Model::subordinate_model().

Referenced by Optimizer::reduce_model().
void objective_reduction ( const Response & full_response, const BoolDeque & sense, const RealVector & full_wts, Response & reduced_response ) const [private]

forward mapping: maps multiple primary response functions to a single weighted objective for single-objective optimizers

This function is responsible for the mapping of multiple objective functions into a single objective for publishing to single-objective optimizers. Used in DOTOptimizer, NPSOLOptimizer, SNLLOptimizer, and SGOPTApplication on every function evaluation. The simple weighting approach (using primaryRespFnWts) is the only technique supported currently. The weightings are used to scale function values, gradients, and Hessians as needed.

References Response::active_set_request_vector(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_value(), Response::function_values(), Response::num_functions(), Minimizer::numConstraints, Minimizer::objective(), Minimizer::objective_gradient(), Minimizer::objective_hessian(), Iterator::outputLevel, and Dakota::write_precision.

Referenced by Optimizer::primary_resp_reducer().

The documentation for this class was generated from the following files:

- DakotaOptimizer.hpp
- DakotaOptimizer.cpp

13.142 OutputManager Class Reference

Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank 0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and delegate to Graphics for X Windows Graphics.

Public Member Functions

- **OutputManager ()**
  
  Default constructor (needed for default environment ctors)

- **OutputManager (const ProgramOptions &prog_opts, int dakota_world_rank=0, bool dakota_mpi_run_flag=false)**
  
  Standard constructor, taking user-specified program options and optionally taking the rank of this process in Dakota’s MPI Comm.

- **~OutputManager ()**
  
  Destructor that closes streams and other outputs.

- **void close_streams ()**
  
  Helper to close streams during destructor or abnormal abort

- **Graphics & graphics ()**
  
  Retrieve the graphics handler object

- **void parse (const ProgramOptions &prog_opts, const ProblemDescDB &problem_db)**
  
  Extract environment options from ProblemDescDB and update from late updates to ProgramOptions.

- **void startup_message (const String &start_msg)**
  
  Set the Dakota startup message ("Running on...")

- **void push_output_tag (const String &iterator_tag, const ProgramOptions &prog_opts, bool force_cout_redirect, bool force_rst_redirect)**
  
  Update the tag to use on files and rebind any streams as needed.

- **String build_output_tag () const**
return the full output tag

- void pop_output_tag()
  (Potentially) remove an output context and rebind streams

- void output_version(std::ostream &os=Cout) const
  Output the current Dakota version.

- void output_startup_message(std::ostream &os=Cout) const
  Output the startup header and time.

- void output_helper(const String &message, std::ostream &os) const
  Output only on Dakota world rank 0 (for version, help, etc.)

- void append_restart(const ParamResponsePair &prp)
  append a parameter/response set to the restart file

- void add_datapoint(const Variables &vars, const String &iface, const Response &response)
  adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model evaluation

- void create_tabular_datastream(const Variables &vars, const Response &resp)
  initialize the tabular datastream on iterator leaders

- void close_tabular()
  close tabular datastream

- void graphics_counter(int cntr)
  set graphicsCntr equal to cntr

- int graphics_counter() const
  return graphicsCntr

- void tabular_counter_label(const std::string &label)
  set tabularCntrLabel equal to label

Public Attributes

- bool graph2DFlag
  whether user requested 2D graphics plots

- bool tabularDataFlag
  whether user requested tabular data file

- bool resultsOutputFlag
  whether user requested results data output

- String tabularDataFile
  filename for tabulation of graphics data

- String resultsOutputFile
  filename for results data

Private Member Functions

- void initial_redirects(const ProgramOptions &prog_opts)
  Perform initial output/error redirects from user requests.

- void read_write_restart(bool restartRequested, bool read_restart_flag, const String &read_restart_filename, size_t stop_restart_eval, const String &write_restart_filename)
  conditionally import evaluations from restart file, then always create or overwrite restart file
Private Attributes

- int worldRank
  output manager handles rank 0 only output when needed
- bool mpirunFlag
  some output is only for MPI runs
- StringArray fileTags
  set of tags for various input/output files (default none)
- bool redirCalled
  temporary variable to prevent recursive tagging initially
- ConsoleRedirector coutRedirector
  set of redirections for Dakota::Cout; stores any tagged filename when there are concurrent Iterators
- ConsoleRedirector cerrRedirector
  set of redirections for Dakota::Cerr; stores any tagged filename when there are concurrent Iterators and error redirection is requested
- std::vector< boost::shared_ptr< RestartWriter > > restartDestinations
  Stack of active restart destinations; end is the last (active) redirection. All remain open until popped or destroyed.
- String startupMessage
  message to print at startup when proceeding to instantiate objects
- Graphics dakotaGraphics
  graphics and tabular data output handler used by meta-iterators, models, and approximations; encapsulated here so destroyed with the OutputManager
- unsigned short tabularFormat
  tabular format options; see enum
- int graphicsCntr
  used for x axis values in 2D graphics and for 1st column in tabular data
- std::ofstream tabularDataFStream
  file stream for tabulation of graphics data within compute_response
- std::string tabularCntrLabel
  label for counter used in first line comment w/i the tabular data file
- short outputLevel
  output level (for debugging only; not passed in)

13.142.1 Detailed Description

Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank 0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and delegate to Graphics for X Windows Graphics.
13.142.2 Constructor & Destructor Documentation

OutputManager ( const ProgramOptions & prog_opts, int dakota_world_rank = 0, bool dakota_mpirun_flag = false )

Standard constructor, taking user-specified program options and optionally taking the rank of this process in Dakota’s MPI Comm.

Only get minimal information off ProgramOptions as may be updated later by broadcast.

References: OutputManager::initial_redirects(), OutputManager::mpirunFlag, and Dakota::start_dakota_heartbeat()

13.142.3 Member Function Documentation

void pop_output_tag ( )

(Potentially) remove an output context and rebind streams

For now this assumes the tag is .<int>

References: OutputManager::build_output_tag(), OutputManager::cerrRedirector, OutputManager::coutRedirector, OutputManager::fileTags, OutputManager::outputLevel, ConsoleRedirector::pop_back(), OutputManager::restartDestinations, and OutputManager::worldRank.

Referenced by ParallelLibrary::pop_output_tag().

void add_datapoint ( const Variables & vars, const String & iface, const Response & response )

adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model evaluation

Adds data to each 2d plot and each tabular data column (one for each active variable and for each response function). graphicsCntr is used for the x axis in the graphics and the first column in the tabular data.

References: Response::active_set_request_vector(), Graphics::add_datapoint(), OutputManager::dakotaGraphics, OutputManager::graphicsCntr, OutputManager::tabularDataFStream, and OutputManager::tabularFormat.

Referenced by Model::evaluate(), Model::synchronize(), and Model::synchronize_nowait().

void create_tabular_datastream ( const Variables & vars, const Response & response )

initialize the tabular datastream on iterator leaders

Opens the tabular data file stream and prints headings, one for each active continuous and discrete variable and one for each response function, using the variable and response function labels. This tabular data is used for post-processing of DAKOTA results in Matlab, Tecplot, etc.

References: OutputManager::build_output_tag(), OutputManager::tabularDataFile, OutputManager::tabularDataFStream, and OutputManager::tabularFormat.

Referenced by SurrBasedMinimizer::initialize_graphics(), and Iterator::initialize_graphics().

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp

13.143 OutputWriter Class Reference

Public Member Functions

- OutputWriter (std::ostream *output_stream)

  ostream constructor; used to construct a writer to existing stream, e.g., std::cout
• **OutputWriter** (const String &output_filename)
  
  *File redirect constructor; opens an overwriting file stream to given name*

• const String & filename () const
  
  *The (possibly empty) file name for this stream*

• std::ostream *output_stream ()
  
  *A pointer to the stream, either cout/cerr or a file*

### Protected Attributes

• String outputFilename
  
  *The name of the output file (empty when constructed from pointer)*

• std::ofstream outputFS
  
  *File output stream for console text; only open if string non-empty*

• std::ostream *outputStream
  
  *Pointer to the stream for this writer*

### 13.143.1 Detailed Description

Component to manage a redirected output or error stream

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp

### 13.144 ParallelConfiguration Class Reference

Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration.

### Public Member Functions

- **ParallelConfiguration ()**
  
  *Default constructor*

- **ParallelConfiguration (const ParallelConfiguration &pl)**
  
  *Copy constructor*

- **~ParallelConfiguration ()**
  
  *Destructor*

- **ParallelConfiguration & operator= (const ParallelConfiguration &pl)**
  
  *Assignment operator*

- const ParallelLevel & w_parallel_level () const
  
  *Return the ParallelLevel corresponding to miPLIter.front()*

- const ParallelLevel & mi_parallel_level (size_t index=_NPOS) const
  
  *Return the ParallelLevel corresponding to miPLIter[index]*

- const ParallelLevel & ie_parallel_level () const
  
  *Return the ParallelLevel corresponding to iePLIter*
• const ParallelLevel & ea_parallel_level () const  
  return the ParallelLevel corresponding to eaPLIter
• bool w_parallel_level_defined () const  
  test for definition of world parallel level
• bool mi_parallel_level_defined (size_t index=_NPOS) const  
  test for definition of meta-iterator-iterator parallel level
• bool ie_parallel_level_defined () const  
  test for definition of iterator-evaluation parallel level
• bool ea_parallel_level_defined () const  
  test for definition of evaluation-analysis parallel level
• ParLevLIter w_parallel_level_iterator () const  
  return miPLIters.front()
• ParLevLIter mi_parallel_level_iterator (size_t index=_NPOS) const  
  return miPLIters[index]
• ParLevLIter ie_parallel_level_iterator () const  
  return iePLIter
• ParLevLIter ea_parallel_level_iterator () const  
  return eaPLIter
• size_t mi_parallel_level_index (ParLevLIter pl_iter) const  
  return the index within miPLIters corresponding to pl_iter
• size_t mi_parallel_level_last_index () const  
  return the index of the last entry in miPLIters

Private Member Functions
• void assign (const ParallelConfiguration &pl)  
  assign the attributes of the incoming pl to this object

Private Attributes
• short numParallelLevels  
  number of parallel levels
• std::vector< ParLevLIter > miPLIters  
  list iterator for world level followed by any concurrent iterator partitions (there may be multiple per parallel configuration instance)
• ParLevLIter iePLIter  
  list iterator identifying the iterator-evaluation parallelLevel (there can only be one)
• ParLevLIter eaPLIter  
  list iterator identifying the evaluation-analysis parallelLevel (there can only be one)
• ParLevLIter endPLIter  
  snapshot of the end of ParallelLibrary::parallelLevels; used for detecting when a component of the parallel configuration has been initialized
13.145. PARALLELDIRECTAPPPLICINTERFACE CLASS REFERENCE

Friends

- class ParallelLibrary

the ParallelLibrary class has special access privileges in order to streamline implementation

13.144.1 Detailed Description

Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration.

Rather than containing the multilevel parallel configuration directly, ParallelConfiguration instead provides a set of list iterators which point into a combined list of ParallelLevels. This approach allows different configurations to reuse ParallelLevels without copying them. A list of ParallelConfigurations is contained in ParallelLibrary (ParallelLibrary::parallelConfigurations).

13.144.2 Member Function Documentation

**const ParallelLevel & mi_parallel_level ( size_t index = NPOS ) const [inline]**

return the ParallelLevel corresponding to miPLIters[index]

If a meaningful index is not provided, return the last mi parallel level. This is useful within the Model context, for which we need the lowest level partition after any meta-iterator recursions.

References Dakota::NPOS, and ParallelConfiguration::miPLIters.

Referenced by ApplicationInterface::set_evaluation_communicators().

**ParLevLIter mi_parallel_level_iterator ( size_t index = NPOS ) const [inline]**

return miPLIters[index]

If a meaningful index is not provided, return the last mi parallel level. This is useful within the Model context, for which we need the lowest level partition after any meta-iterator recursions.

References Dakota::NPOS, and ParallelConfiguration::miPLIters.

The documentation for this class was generated from the following file:

- ParallelLibrary.hpp

13.145 ParallelDirectApplicInterface Class Reference

Sample derived interface class for testing parallel simulator plug-ins using assign_rep().

Inheritance diagram for ParallelDirectApplicInterface:

```
Interface

ApplicationInterface

DirectApplicInterface

ParallelDirectApplicInterface
```
Public Member Functions

- **ParallelDirectApplicInterface** (const Dakota::ProblemDescDB &problem_db, const MPI_Comm &analysis_comm)
  - constructor
- **~ParallelDirectApplicInterface** ()
  - destructor

Protected Member Functions

- int **derived_map_ac** (const Dakota::String &ac_name)
  - execute an analysis code portion of a direct evaluation invocation
- void **derived_map_asynch** (const Dakota::ParamResponsePair &pair)
  - no-op hides base error; job batching occurs within wait_local_evaluations()
- void **wait_local_evaluations** (Dakota::PRPQueue &prp_queue)
  - evaluate the batch of jobs contained in prp_queue
- void **test_local_evaluations** (Dakota::PRPQueue &prp_queue)
  - invokes wait_local_evaluations() (no special nowait support)
- void **set_communicators_checks** (int max_eval_concurrency)
  - no-op hides default run-time error checks at DirectApplicInterface level

Private Member Functions

- int **text_book** (const Dakota::RealVector &c_vars, const Dakota::ShortArray &asv, Dakota::RealVector &fn_vals, Dakota::RealMatrix &fn_grads, Dakota::RealSymMatrixArray &fn_hessians)
  - demo evaluator function for parallel plug-ins

Additional Inherited Members

13.145.1 Detailed Description

Sample derived interface class for testing parallel simulator plug-ins using assign_rep().

The plug-in ParallelDirectApplicInterface resides in namespace SIM and uses a copy of textbook() to perform parallel parameter to response mappings. It is used to demonstrate plugging in a parallel direct analysis driver into Dakota in library mode. Test input files can then use an analysis_driver of "plugin_textbook".

13.145.2 Member Function Documentation

void **test_local_evaluations** ( Dakota::PRPQueue & prp_queue )  [inline], [protected]

invokes wait_local_evaluations() (no special nowait support)

For use by ApplicationInterface::serve_evaluations_asynch(), which can provide a batch processing capability within message passing schedulers (called using chain ApplicationInterface::serve_evaluations() from Model::serve() from IteratorScheduler::run_iterator()).

References ParallelDirectApplicInterface::wait_local_evaluations().

The documentation for this class was generated from the following files:

- PluginParallelDirectApplicInterface.hpp
- PluginParallelDirectApplicInterface.cpp
13.146 ParallelLevel Class Reference

Container class for the data associated with a single level of communicator partitioning.

Public Member Functions

- `ParallelLevel ()`
  *default constructor*

- `ParallelLevel (const ParallelLevel &pl)`
  *copy constructor*

- `~ParallelLevel ()`
  *destructor*

- `ParallelLevel & operator= (const ParallelLevel &pl)`
  *assignment operator*

- `bool dedicated_master () const`
  *return dedicatedMasterFlag*

- `bool server_master () const`
  *return serverMasterFlag*

- `bool message_pass () const`
  *return messagePass*

- `bool idle_partition () const`
  *return idlePartition*

- `int num_servers () const`
  *return numServers*

- `int processors_per_server () const`
  *return procsPerServer*

- `int processor_remainder () const`
  *return procRemainder*

- `const MPI_Comm & server_intra_communicator () const`
  *return serverIntraComm*

- `int server_communicator_rank () const`
  *return serverCommRank*

- `int server_communicator_size () const`
  *return serverCommSize*

- `const MPI_Comm & hub_server_intra_communicator () const`
  *return hubServerIntraComm*

- `int hub_server_communicator_rank () const`
  *return hubServerCommRank*

- `int hub_server_communicator_size () const`
  *return hubServerCommSize*

- `const MPI_Comm & hub_server_inter_communicator () const`
  *return hubServerInterComm*

- `MPI_Comm * hub_server_inter_communicators () const`
  *return hubServerInterComms*
CHAPTER 13. CLASS DOCUMENTATION

- int server_id () const
  
  return serverId

- void read (MPIUnpackBuffer &s)
  
  read a ParallelLevel object from a packed MPI buffer

- void write (MPIPackBuffer &s) const
  
  write a ParallelLevel object to a packed MPI buffer

- bool null (const MPI_Comm &comm)
  
  test comm for MPI_COMM_NULL

- bool special (const MPI_Comm &comm)
  
  test comm for special identity that cannot be deallocated

- void clear ()
  
  deallocate the communicators in this ParallelLevel

- void alias (const ParallelLevel &pl)
  
  assign the attributes of the incoming pl to this object. For communicators, this is a lightweight copy which assigns the same pointer values as the incoming pl, resulting in the same context.

- void copy (const ParallelLevel &pl)
  
  deep copy the attributes of the incoming pl to this object using MPI_Comm_dup to create equivalent communicators with a unique context.

- void copy_config (const ParallelLevel &pl)
  
  copy the scalar attributes of the incoming pl to this object, omitting communicators

Private Attributes

- bool ownCommFlag
  
  signals Comm ownership for deallocation

- bool dedicatedMasterFlag
  
  signals dedicated master partitioning

- bool commSplitFlag
  
  signals a communicator split was used

- bool serverMasterFlag
  
  identifies master server processors

- bool messagePass
  
  flag for message passing at this level, < indicating work assignment among servers

- bool idlePartition
  
  identifies presence of an idle processor < partition at this level

- int numServers
  
  number of servers

- int procsPerServer
  
  processors per server

- int procRemainder
  
  proc remainder after equal distribution

- int serverId
server identifier

- MPI_Comm serverIntraComm
  intracomm. for each server partition
- int serverCommRank
  rank in serverIntraComm
- int serverCommSize
  size of serverIntraComm
- MPI_Comm hubServerIntraComm
  intracomm for all serverCommRank==0
  < w/i next higher level serverIntraComm
- int hubServerCommRank
  rank in hubServerIntraComm
- int hubServerCommSize
  size of hubServerIntraComm
- MPI_Comm hubServerInterComm
  intercomm. between a server & the hub
  < (on server partitions only)
- MPI_Comm * hubServerInterComms
  intercomm. array on hub processor

Friends

- class ParallelLibrary
  the ParallelLibrary class has special access privileges in order to streamline implementation

13.146.1 Detailed Description

Container class for the data associated with a single level of communicator partitioning.
A list of these levels is contained in ParallelLibrary (ParallelLibrary::parallelLevels), which defines all of the parallelism levels across one or more multilevel parallelism configurations.

13.146.2 Member Function Documentation

void clear() [inline]
deallocate the communicators in this ParallelLevel

This appears to be more robust outside of the destructor due to interactions among managed deallocation and default deallocation (e.g., explicitly freeing a communicator and then default deallocating its handle).

References ParallelLevel::dedicatedMasterFlag, ParallelLevel::hubServerInterComm, ParallelLevel::hubServerInterComms, ParallelLevel::hubServerIntraComm, ParallelLevel::idlePartition, ParallelLevel::numServers, ParallelLevel::ownCommFlag, ParallelLevel::serverId, ParallelLevel::serverIntraComm, and ParallelLevel::special().

The documentation for this class was generated from the following file:

- ParallelLibrary.hpp

13.147 ParallelLibrary Class Reference

Class for partitioning multiple levels of parallelism and managing message passing within these levels.
Public Member Functions

- **ParallelLibrary ()**
  - default constructor (used for dummy_lib)
- **ParallelLibrary (const MPIManager &mpi_mgr, ProgramOptions &prog_opts, OutputManager &output_mgr)**
  - stand-alone and default library mode constructor; don’t require options
- **∼ParallelLibrary ()**
  - destructor
- **const ParallelLevel & init_iterator_communicators (int iterator_servers, int procs_per_iterator, int min_procs_per_iterator, int max_procs_per_iterator, int min_iterator_concurrency, short default_config, short iterator_scheduling, bool peer_dynamic_avail)**
  - split MPI_COMM_WORLD into iterator communicators
- **const ParallelLevel & init_evaluation_communicators (int evaluation_servers, int procs_per_evaluation, int min_procs_per_eval, int max_procs_per_eval, int max_evaluation_concurrency, int asynch_local_evaluation_concurrency, short default_config, short evaluation_scheduling, bool peer_dynamic_avail)**
  - split an iterator communicator into evaluation communicators
- **const ParallelLevel & init_analysis_communicators (int analysis_servers, int procs_per_analysis, int min_procs_per_analysis, int max_procs_per_analysis, int max_analysis_concurrency, int asynch_local_analysis_concurrency, short default_config, short analysis_scheduling, bool peer_dynamic_avail)**
  - split an evaluation communicator into analysis communicators
- **void print_configuration ()**
  - print the parallel level settings for a particular parallel configuration
- **void push_output_tag (const ParallelLevel &pl)**
  - conditionally append an iterator server id tag to the hierarchical output tag, manage restart, and rebind cout/cerr
- **void pop_output_tag (const ParallelLevel &pl)**
  - pop the last output tag and rebind streams as needed; pl isn’t yet used, but may be in the future when we generalize to arbitrary output context switching
- **void write_restart (const ParamResponsePair &prp)**
  - write a parameter/response set to the restart file
- **ProgramOptions & program_options ()**
  - return programOptions reference
- **OutputManager & output_manager ()**
  - return outputManager reference
- **void terminate_modelcenter ()**
  - terminate ModelCenter if running
- **void abort_helper (int code)**
  - finalize MPI with correct communicator for abort
- **bool command_line_check () const**
  - return checkFlag
- **bool command_line_pre_run () const**
  - return preRunFlag
- **bool command_line_run () const**
  - return runFlag
- **bool command_line_post_run () const**
return postRunFlag
• bool command_line_user_modes () const
  return userModesFlag
• const String & command_line_pre_run_input () const
  preRunInput filename
• const String & command_line_pre_run_output () const
  preRunOutput filename
• const String & command_line_run_input () const
  runInput filename
• const String & command_line_run_output () const
  runOutput filename
• const String & command_line_post_run_input () const
  postRunInput filename
• const String & command_line_post_run_output () const
  postRunOutput filename
• void send (MPIPackBuffer &send_buff, int dest, int tag, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking buffer send at the current communication level
• void send (int &send_int, int dest, int tag, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking integer send at the current communication level
• void isend (MPIPackBuffer &send_buff, int dest, int tag, MPI Request &send_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking buffer send at the current communication level
• void isend (int &send_int, int dest, int tag, MPI Request &send_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking integer send at the current communication level
• void recv (MPIUnpackBuffer &recv_buff, int source, int tag, MPI Status &status, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking buffer receive at the current communication level
• void recv (int &recv_int, int source, int tag, MPI Status &status, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking integer receive at the current communication level
• void irecv (MPIUnpackBuffer &recv_buff, int source, int tag, MPI Request &recv_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking buffer receive at the current communication level
• void irecv (int &recv_int, int source, int tag, MPI Request &recv_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking integer receive at the current communication level
• void check_mi_index (size_t &index) const
  process _NPOS default and perform error checks
• void send_mi (int &send_int, int dest, int tag, size_t index=_NPOS)
  blocking send at the metaiterator-iterator communication level
• void isend_mi (int &send_int, int dest, int tag, MPI Request &send_req, size_t index=_NPOS)
  nonblocking send at the metaiterator-iterator communication level
• void recv_mi (int &recv_int, int source, int tag, MPI_Status &status, size_t index=NPOS)
  
  blocking receive at the metaiterator-iterator communication level

• void irecv_mi (int &recv_int, int source, int tag, MPI_Request &recv_req, size_t index=NPOS)
  
  nonblocking receive at the metaiterator-iterator communication level

• void send_mi (MPIPackBuffer &send_buff, int dest, int tag, size_t index=NPOS)
  
  blocking send at the metaiterator-iterator communication level

• void isend_mi (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req, size_t index=NPOS)
  
  nonblocking send at the metaiterator-iterator communication level

• void recv_mi (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status, size_t index=NPOS)
  
  blocking receive at the metaiterator-iterator communication level

• void irecv_mi (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req, size_t index=-NPOS)
  
  nonblocking receive at the metaiterator-iterator communication level

• void send_ie (int &send_int, int dest, int tag)
  
  blocking send at the iterator-evaluation communication level

• void isend_ie (int &send_int, int dest, int tag, MPI_Request &send_req)
  
  nonblocking send at the iterator-evaluation communication level

• void recv_ie (int &recv_int, int source, int tag, MPI_Status &status)
  
  blocking receive at the iterator-evaluation communication level

• void irecv_ie (int &recv_int, int source, int tag, MPI_Request &recv_req)
  
  nonblocking receive at the iterator-evaluation communication level

• void send_ie (MPIPackBuffer &send_buff, int dest, int tag)
  
  blocking send at the iterator-evaluation communication level

• void isend_ie (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req)
  
  nonblocking send at the iterator-evaluation communication level

• void recv_ie (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status)
  
  blocking receive at the iterator-evaluation communication level

• void irecv_ie (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req)
  
  nonblocking receive at the iterator-evaluation communication level

• void send_ea (int &send_int, int dest, int tag)
  
  blocking send at the evaluation-analysis communication level

• void isend_ea (int &send_int, int dest, int tag, MPI_Request &send_req)
  
  nonblocking send at the evaluation-analysis communication level

• void recv_ea (int &recv_int, int source, int tag, MPI_Status &status)
  
  blocking receive at the evaluation-analysis communication level

• void irecv_ea (int &recv_int, int source, int tag, MPI_Request &recv_req)
  
  nonblocking receive at the evaluation-analysis communication level

• void bcast (int &data, const ParallelLevel &pl)
  broadcast an integer across the serverIntraComm of a ParallelLevel

• void bcast (short &data, const ParallelLevel &pl)
  broadcast an integer across the serverIntraComm of a ParallelLevel

• void bcast (MPIPackBuffer &send_buff, const ParallelLevel &pl)
  broadcast a MPIPackBuffer across the serverIntraComm of a ParallelLevel
• void bcast (MPIUnpackBuffer &recv_buff, const ParallelLevel &pl)
  broadcast a MPIUnpackBuffer across the serverIntraComm of a ParallelLevel
• void bcast hs (int &data, const ParallelLevel &pl)
  broadcast an integer across the hubServerIntraComm of a ParallelLevel
• void bcast hs (MPIPackBuffer &send_buff, const ParallelLevel &pl)
  broadcast a MPIPackBuffer across the hubServerIntraComm of a ParallelLevel
• void bcast hs (MPIUnpackBuffer &recv_buff, const ParallelLevel &pl)
  broadcast a MPIUnpackBuffer across the hubServerIntraComm of a ParallelLevel
• void bcast (MPIPackBuffer &send_buff)
  broadcast a packed buffer across MPI COMM WORLD
• void bcast i (int &data, size_t index=NPOS)
  broadcast an integer across an iterator communicator
• void bcast i (short &data, size_t index=NPOS)
  broadcast a short integer across an iterator communicator
• void bcast a (int &data)
  broadcast an integer across an analysis communicator
• void bcast mi (int &data, size_t index=NPOS)
  broadcast an integer across a metaiterator-iterator intra communicator
• void bcast (MPIPackBuffer &send_buff)
  broadcast a packed buffer across MPI COMM WORLD
• void bcast i (MPIPackBuffer &send_buff, size_t index=NPOS)
  broadcast a packed buffer across an iterator communicator
• void bcast e (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an evaluation communicator
• void bcast a (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an analysis communicator
• void bcast mi (MPIPackBuffer &send_buff, size_t index=NPOS)
  broadcast a packed buffer across a metaiterator-iterator intra communicator
• void bcast (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer broadcast across MPI COMM WORLD
• void bcast i (MPIUnpackBuffer &recv_buff, size_t index=NPOS)
  matching receive for packed buffer bcast across an iterator communicator
• void bcast e (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an evaluation communicator
• void bcast a (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an analysis communicator
• void bcast mi (MPIUnpackBuffer &recv_buff, size_t index=NPOS)
  matching recv for packed buffer bcast across a metaiterator-iterator intra comm
• void barrier w ()
  enforce MPI_Barrier on MPI COMM WORLD
• void barrier i (size_t index=NPOS)
enforce MPI_Barrier on an iterator communicator

- void barrier_e ()
  enforce MPI_Barrier on an evaluation communicator

- void barrier_a ()
  enforce MPI_Barrier on an analysis communicator

- void reduce_sum_ea (double *local_vals, double *sum_vals, int num_vals)
  compute a sum over an eval-analysis intra-communicator using MPI_Reduce

- void reduce_sum_a (double *local_vals, double *sum_vals, int num_vals)
  compute a sum over an analysis communicator using MPI_Reduce

- void test (MPI_Request &request, int &test_flag, MPI_Status &status)
  test a nonblocking send/receive request for completion

- void wait (MPI_Request &request, MPI_Status &status)
  wait for a nonblocking send/receive request to complete

- void waitall (int num_recs, MPI_Request *&recv_reqs)
  wait for all messages from a series of nonblocking receives

- void waitsome (int num_sends, MPI_Request *&recv_requests, int &num_recs, int *&index_array, MPI_Status *&status_array)
  wait for at least one message from a series of nonblocking receives but complete all that are available

- void free (MPI_Request &request)
  free an MPI_Request

- int world_size () const
  return MPIManager::worldSize

- int world_rank () const
  return MPIManager::worldRank

- bool mpirun_flag () const
  return MPIManager::mpirunFlag

- bool is_null () const
  return dummyFlag

- Real parallel_time () const
  returns current MPI wall clock time

- void parallel_configuration_iterator (ParConfigLIter pc_iter)
  set the current ParallelConfiguration node

- ParConfigLIter parallel_configuration_iterator () const
  return the current ParallelConfiguration node

- const ParallelConfiguration & parallel_configuration () const
  return the current ParallelConfiguration instance

- size_t num_parallel_configurations () const
  returns the number of entries in parallelConfigurations

- bool parallel_configuration_is_complete ()
  identifies if the current ParallelConfiguration has been fully populated

- void increment_parallel_configuration (ParLevLIter mi_pl_iter)
  add a new node to parallelConfigurations and increment currPCIter; limit miPLIIter within new configuration to mi_pl_iter level
void increment_parallel_configuration ()
add a new node to parallelConfigurations and increment currPCIter; copy all of miPLIterators into new configuration

bool w_parallel_level_defined () const

test current parallel configuration for definition of world parallel level

bool mi_parallel_level_defined (size_t index=_NPOS) const

test current parallel configuration for definition of meta-iterator-iterator parallel level

bool ie_parallel_level_defined () const

test current parallel configuration for definition of iterator-evaluation parallel level

bool ea_parallel_level_defined () const

test current parallel configuration for definition of evaluation-analysis parallel level

ParLevLIter w_parallel_level_iterator ()
for this level, access through ParallelConfiguration is not necessary

size_t parallel_level_index (ParLevLIter pl_iter)
return the index within parallelLevels corresponding to pl_iter

std::vector< MPIComm > analysis_intra_communicators ()
return the set of analysis intra communicators for all parallel configurations (used for setting up direct simulation interfaces prior to execution time).

Private Member Functions

void init_mpi_comm ()
convenience function for initializing DAKOTA’s top-level MPI communicators, based on dakotaMPIComm

void initialize_timers ()
initialize DAKOTA and UTILIB timers

void output_timers ()
conditionally output timers in destructor

void init_communicators (const ParallelLevel &parent_pl, int num_servers, int procs_per_server, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int asynchronous_local_concurrency, short default_config, short scheduling_override, bool peer_dynamic_avail)
split a parent communicator into child server communicators

void split_communicator_dedicated_master (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
split a parent communicator into a dedicated master processor and num_servers child communicators

void split_communicator_peer_partition (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
split a parent communicator into num_servers peer child communicators (no dedicated master processor)

void resolve_inputs (ParallelLevel &child_pl, int avail_procs, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int capacity_multiplier, short default_config, short scheduling_override, bool peer_dynamic_avail, bool print_rank)
resolve user inputs into a sensible partitioning scheme

void bcast (int &data, const MPI_Comm &comm)
broadcast an integer across a communicator

void bcast (short &data, const MPI_Comm &comm)
broadcast a short integer across a communicator

void bcast (MPIPackBuffer &send_buff, const MPI_Comm &comm)
send a packed buffer across a communicator using a broadcast
CHAPTER 13. CLASS DOCUMENTATION

- `void bcast (MPIUnpackBuffer &recv_buff, const MPI_Comm &comm)`
  matching receive for a packed buffer broadcast

- `void barrier (const MPI_Comm &comm)`
  enforce MPI_Barrier on comm

- `void reduce_sum (double *local_vals, double *sum_vals, int num_vals, const MPI_Comm &comm)`
  compute a sum over comm using MPI_Reduce

- `void check_error (const String &err_source, int err_code)`
  check the MPI return code and abort if error

- `void alias_as_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)`
  convenience function for updating child serverIntraComm from parent serverIntraComm (shallow Comm copy)

- `void copy_as_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)`
  convenience function for updating child serverIntraComm from parent serverIntraComm (deep Comm copy)

- `void alias_as_hub_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)`
  convenience function for updating child hubServerIntraComm from parent serverIntraComm (shallow Comm copy)

- `void copy_as_hub_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)`
  convenience function for updating child hubServerIntraComm from parent serverIntraComm (deep Comm copy)

Private Attributes

- `const MPIManager & mpiManager`
  reference to the MPI manager with Dakota’s MPI options

- `ProgramOptions & programOptions`
  programOptions is non-const due to updates from broadcast

- `OutputManager & outputManager`
  Non-const output handler to help with file redirection.

- `bool dummyFlag`
  prevents multiple MPI_Finalize calls due to dummy_lib

- `bool outputTimings`
  timing info only beyond help/version/check

- `Real startCPUPTime`
  start reference for UTILIB CPU timer

- `Real startWCTime`
  start reference for UTILIB wall clock timer

- `Real startMPITime`
  start reference for MPI wall clock timer

- `long startClock`
  start reference for local clock() timer measuring < parent+child CPU

- `std::list< ParallelLevel > parallelLevels`
  the complete set of parallelism levels for managing multilevel parallelism among one or more configurations

- `std::list< ParallelConfiguration > parallelConfigurations`
  the set of parallel configurations which manage list iterators for indexing into parallelLevels

- `ParConfigLIter currPCIter`
  list iterator identifying the current node in parallelConfigurations
13.147. PARALLELLIBRARY CLASS REFERENCE

13.147.1 Detailed Description

Class for partitioning multiple levels of parallelism and managing message passing within these levels.

The ParallelLibrary class encapsulates all of the details of performing message passing within multiple levels of parallelism. It provides functions for partitioning of levels according to user configuration input and functions for passing messages within and across MPI communicators for each of the parallelism levels. If support for other message-passing libraries beyond MPI becomes needed (PVM, ...), then ParallelLibrary would be promoted to a base class with virtual functions to encapsulate the library-specific syntax.

13.147.2 Constructor & Destructor Documentation

ParallelLibrary ( )

default constructor (used for dummy_lib)

This constructor is used for creation of the global dummy_lib object, which is used to satisfy initialization requirements when the real ParallelLibrary object is not available.

ParallelLibrary ( const MPIManager & mpi_mgr, ProgramOptions & prog_opts, OutputManager & output mgr )

stand-alone and default library mode constructor; don’t require options

library mode constructor accepting communicator

TODO: Update comment.

Same constructor is used for executable and library environments and sequencing of object construction is ordered, so no need to separately get updates off command line (programOptions)

References ParallelLibrary::init_mpi_comm(), and ParallelLibrary::initialize_timers().

13.147.3 Member Function Documentation

void push_output_tag ( const ParallelLevel & pl )

conditionally append an iterator server id tag to the hierarchical output tag, manage restart, and rebind cout/cerr

If the user has specified the use of files for DAKOTA standard output and/or standard error, then bind these filenames to the Cout/Cerr macros. In addition, if concurrent iterators are to be used, create and tag multiple output streams in order to prevent jumbled output. Manage restart file(s) by processing any incoming evaluations from an old restart file and by setting up the binary output stream for new evaluations. Only master iterator processor(s) read & write restart information. This function must follow init_iterator_communicators so that restart can be managed properly for concurrent iterator strategies. In the case of concurrent iterators, each iterator has its own restart file tagged with iterator number.

References ParallelLibrary::bcast(), ParallelLevel::dedicatedMasterFlag, OutputManager::graph2DFlag, ParallelLevel::hubServerCommRank, ParallelLevel::hubServerCommSize, ParallelLevel::hubServerIntraComm, ParallelLevel::numServers, ParallelLibrary::outputManager, ParallelLibrary::programOptions, OutputManager::push_output_tag(), OutputManager::resultsOutputFile, OutputManager::resultsOutputFlag, ParallelLevel::serverCommRank, ParallelLevel::serverId, MPIPackBuffer::size(), OutputManager::tabularDataFile, and OutputManager::tabularDataFlag.

Referenced by Environment::construct(), and IteratorScheduler::partition().

void terminate_modelcenter ( )

terminate ModelCenter if running
Close streams associated with manage_outputs and manage_restart and terminate any additional services that may be active.

References Dakota::abort_handler(), Dakota::dc_ptr_int, and Dakota::mc_ptr_int.

Referenced by ParallelLibrary::~ParallelLibrary().

```cpp
void increment_parallel_configuration ( ParLevIter mi_pl_iter ) [inline]
```

add a new node to parallelConfigurations and increment currPCIter; limit miPLIterers within new configuration to mi_pl_iter level

Called from the ParallelLibrary ctor and from Model::init_communicators(). An increment is performed for each Model initialization except the first (which inherits the world level from the first partial configuration).

References ParallelLibrary::currPCIter, ParallelConfiguration::eaPLIter, ParallelConfiguration::endPLIter, ParallelConfiguration::iePLIter, ParallelConfiguration::mpILIterers, ParallelConfiguration::numParallelLevels, ParallelLibrary::parallelConfigurations, and ParallelLibrary::parallelLevels.

Referenced by Iterator::init_communicators(), and Model::init_communicators().

```cpp
void init_mpi_comm ( ) [private]
```

convenience function for initializing DAKOTA's top-level MPI communicators, based on dakotaMPIComm

shared function for initializing based on passed MPI_Comm

References Dakota::abort_handler(), MPIManager::dakota_mpi_comm(), ParallelLibrary::increment_parallel_configuration(), ParallelLibrary::mpiManager, MPIManager::mpirun_flag(), ParallelLevel::numServers, ParallelLibrary::outputManager, ParallelLibrary::parallelLevels, ParallelLevel::procsPerServer, ParallelLevel::serverCommRank, ParallelLevel::serverCommSize, ParallelLevel::serverId, ParallelLevel::serverIntraComm, ParallelLevel::serverMasterFlag, ParallelLibrary::startMPTime, OutputManager::startup_message(), MPIManager::world_rank(), and MPIManager::world_size().

Referenced by ParallelLibrary::ParallelLibrary().

```cpp
void init_communicators ( const ParallelLevel & parent_pl, int num_servers, int procs_per_server, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int async_local_concurrency, short default_config, short scheduling_override, bool peer_dynamic_avail ) [private]
```

split a parent communicator into child server communicators

Split parent communicator into concurrent child server partitions as specified by the passed parameters. This constructs new child intra-communicators and parent-child inter-communicators. This fn is called from Meta-Iterators and NestedModel for the concurrent iterator level and from ApplicationInterface::init_communicators() for the concurrent evaluation and concurrent analysis levels.

References ParallelLibrary::currPCIter, ParallelLevel::dedicatedMasterFlag, ParallelLevel::messagePass, ParallelLevel::numServers, ParallelLibrary::parallelLevels, ParallelLevel::procsPerServer, ParallelLibrary::resolve_inputs(), ParallelLevel::serverCommRank, ParallelLevel::serverCommSize, ParallelLibrary::split_communicator_dedicated_master(), and ParallelLibrary::split_communicator_peer_partition().

Referenced by ParallelLibrary::init_analysis_communicators(), ParallelLibrary::init_evaluation_communicators(), and ParallelLibrary::init_iterator_communicators().

```cpp
void resolve_inputs ( ParallelLevel & child_pl, int avail_procs, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int capacity_multiplier, short default_config, short scheduling_override, bool peer_dynamic_avail, bool print_rank ) [private]
```

resolve user inputs into a sensible partitioning scheme
This function is responsible for the "auto-configure" intelligence of DAKOTA. It resolves a variety of inputs and overrides into a sensible partitioning configuration for a particular parallelism level. It also handles the general case in which a user’s specification request does not divide out evenly with the number of available processors for the level. If num_servers & procs_per_server are both nondefault, then the former takes precedence.

References Dakota::abort_handler(), ParallelLevel::dedicatedMasterFlag, ParallelLevel::numServers, ParallelLevel::procRemainder, and ParallelLevel::procsPerServer.

Referenced by ParallelLibrary::init_communicators().

The documentation for this class was generated from the following files:

- ParallelLibrary.hpp
- ParallelLibrary.cpp

### 13.148 ParamResponsePair Class Reference

Container class for a variables object, a response object, and an evaluation id.

#### Public Member Functions

- **ParamResponsePair ()**
  
  default constructor

- **ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, bool deep_copy=false)**
  
  alternate constructor for temporaries

- **ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, const int eval_id, bool deep_copy=true)**
  
  standard constructor for history uses

- **ParamResponsePair (const ParamResponsePair &pair)**
  
  copy constructor

- **~ParamResponsePair ()**
  
  destructor

- **ParamResponsePair & operator= (const ParamResponsePair &pair)**
  
  assignment operator

- **void read (std::istream &s)**
  
  read a ParamResponsePair object from an std::istream

- **void write (std::ostream &s) const**
  
  write a ParamResponsePair object to an std::ostream

- **void read.annotated (std::istream &s)**
  
  read a ParamResponsePair object in annotated format from an std::istream

- **void write.annotated (std::ostream &s) const**
  
  write a ParamResponsePair object in annotated format to an std::ostream

- **void write.tabular (std::ostream &s, unsigned short tabular_format) const**
  
  write a ParamResponsePair object in tabular format (all variables active/inactive) to an std::ostream

- **void write.tabular.labels (std::ostream &s, unsigned short tabular_format) const**
  
  write PRP labels in tabular format to an std::ostream

- **void read (MPIUnpackBuffer &s)**
read a ParamResponsePair object from a packed MPI buffer

- void write (MPIPackBuffer &s) const
  - write a ParamResponsePair object to a packed MPI buffer

- int eval_id () const
  - return the evaluation identifier

- void eval_id (int id)
  - set the evaluation identifier

- const String & interface_id () const
  - return the interface identifier from evalInterfaceIds

- void interface_id (const String &id)
  - set the interface identifier within evalInterfaceIds

- const IntStringPair & eval_interface_ids () const
  - return the aggregate eval/interface identifier from the response object

- const Variables & variables () const
  - return the parameters object

- void variables (const Variables &vars)
  - set the parameters object

- const Response & response () const
  - return the response object

- void response (const Response &resp)
  - set the response object

- const ActiveSet & active_set () const
  - return the active set object from the response object

- void active_set (const ActiveSet &set)
  - set the active set object within the response object

Private Member Functions

- template<class Archive >
  - void serialize (Archive &ar, const unsigned int version)
    - serialize the PRP: write and read are symmetric for this class

Private Attributes

- Variables prpVariables
  - the set of parameters for the function evaluation

- Response prpResponse
  - the response set for the function evaluation

- IntStringPair evalInterfaceIds
  - the evalInterfaceIds aggregate
Friends

- class boost::serialization::access
  
  allow boost access to serialize this class

- bool operator==(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  
  equality operator

- bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  
  inequality operator

13.148.1 Detailed Description

Container class for a variables object, a response object, and an evaluation id.

ParamResponsePair provides a container class for association of the input for a particular function evaluation (a variables object) with the output from this function evaluation (a response object), along with an evaluation identifier. This container defines the basic unit used in the data pairs cache, in restart file operations, and in a variety of scheduling algorithm queues. With the advent of STL, replacement of arrays of this class with map<> and pair<> template constructs may be possible (using map<pair<int,String>, pair<Variables,Response>>, for example), assuming that deep copies, I/O, alternate constructors, etc., can be adequately addressed. Boost tuple<> may also be a candidate.

13.148.2 Constructor & Destructor Documentation

ParamResponsePair (const Variables & vars, const String & interface_id, const Response & response, bool deep_copy = false) [inline]

alternate constructor for temporaries

Uses of this constructor often employ the standard Variables and Response copy constructors to share representations since this constructor is commonly used for search pairs (which are local instantiations that go out of scope prior to any changes to values; i.e., they are not used for history).

ParamResponsePair (const Variables & vars, const String & interface_id, const Response & response, const int eval_id, bool deep_copy = true) [inline]

standard constructor for history uses

Uses of this constructor often do not share representations since deep copies are used when history mechanisms (e.g., data_pairs and beforeSynchCorePRPQueue) are involved.

13.148.3 Member Function Documentation

void read ( MPIUnpackBuffer & s ) [inline]

read a ParamResponsePair object from a packed MPI buffer

interfaceId is omitted since master processor retains interface ids and communicates asv and response data only with slaves.

References ParamResponsePair::evalInterfaceIds, ParamResponsePair::prpResponse, and ParamResponsePair::prpVariables.
void write( MPIPackBuffer & s ) const  [inline]

write a ParamResponsePair object to a packed MPI buffer.
interfaceId is omitted since master processor retains interface ids and communicates asv and response data only with slaves.
References ParamResponsePair::evalInterfaceIds, ParamResponsePair::prpResponse, and ParamResponsePair::prpVariables.

13.148.4 Member Data Documentation

IntStringPair evalInterfaceIds  [private]

the evalInterfaceIds aggregate

the function evaluation identifier (assigned from Interface::evalIdCntr) is paired with the interface used to generate the response object. Used in PRPCache id_vars_set_compare to prevent duplicate detection on results from different interfaces. evalInterfaceIds belongs here rather than in Response since some Response objects involve consolidation of several fn evals (e.g., Model::synchronize_derivatives()) that are not, in total, generated by a single interface. The prPair, on the other hand, is used for storage of all low level fn evals that get evaluated in ApplicationInterface::map().

Referenced by ParamResponsePair::eval_ids(), ParamResponsePair::eval_interface_ids(), ParamResponsePair::interface_id(), ParamResponsePair::operator=(), Dakota::operator==(), ParamResponsePair::read(), and ParamResponsePair::write().

The documentation for this class was generated from the following file:

- ParamResponsePair.hpp

13.149 ParamStudy Class Reference

Class for vector, list, centered, and multidimensional parameter studies.

Inheritance diagram for ParamStudy:

```
    Iterator
     |      
     |      
    Analyzer
     |      
     |      
PStudyDACE
     |      
     |      
    ParamStudy
```

Public Member Functions

- **ParamStudy (ProblemDescDB &problem_db, Model &model)**
  
  *constructor*

- **~ParamStudy ()**

  *destructor*

- **bool resize ()**
reinitializes iterator based on new variable size

- **void** pre_run()
  
  *pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- **void** core_run()
  
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

- **void** post_input()
  
  *read tabular data for post-run mode*

- **void** post_run(std::ostream &s)
  
  *post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way*

### Private Member Functions

- **void** sample()
  
  *performs the parameter study by sampling from a list of points*

- **void** vector_loop()
  
  *performs the parameter study by sampling along a vector, starting from an initial point followed by numSteps increments along continuous/discrete step vectors*

- **void** centered_loop()
  
  *performs a number of plus and minus offsets for each parameter centered about an initial point*

- **void** multidim_loop()
  
  *performs a full factorial combination for all intersections defined by a set of multidimensional partitions*

- **bool** load_distribute_points(const String &points_filename, unsigned short tabular_format, bool active_only)
  
  *load list of points from data file and distribute among listCVPoints, listDIVPoints, listDSVPoints, and listDRVPoints*

- **template<typename OrdinalType, typename ScalarTypeA, typename ScalarTypeC, typename ScalarTypeDI, typename ScalarTypeDS, typename ScalarTypeDR>**
  
  **bool** distribute(const Teuchos::SerialDenseVector<OrdinalType, ScalarTypeA> &all_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeC> &c_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDI> &di_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDS> &ds_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDR> &dr_data)
  
  *distributes incoming all vector in standard variable ordering among continuous, discrete int, discrete string, and discrete real vectors*

- **template<typename ScalarType>**
  
  **bool** distribute(const std::vector<ScalarType> &all_data, std::vector<ScalarType> &c_data, std::vector<ScalarType> &di_data, std::vector<ScalarType> &ds_data, std::vector<ScalarType> &dr_data)
  
  *distributes incoming all array in standard variable ordering among continuous, discrete int, discrete string, and discrete real arrays*

- **bool** distribute_list_of_points(const RealVector &list_of_pts)
  
  *distributes list of pts coming from user spec among listCVPoints, listDIVPoints, listDSVPoints, and listDRVPoints*

- **void** final_point_to_step_vector()
  
  *compute step vectors from finalPoint, initial points, and numSteps*

- **void** distribute_partitions()
  
  *compute step vectors from (cont, discInt, discString, discReal) VarPartitions and global bounds*

- **bool** check_num_steps(int num_steps)
perform error checks on numSteps

- bool check_step_vector (const RealVector &step_vector)

perform error checks on numSteps

- bool check_final_point (const RealVector &final_pt)

perform error checks on finalPoint

- bool check_steps_per_variable (const IntVector &steps_per_var)

perform error checks on stepsPerVariable

- bool check_variable_partitions (const UShortArray &partitions)

perform error checks on variable partitions

- bool check_finite_bounds ()

check for finite variable bounds within iteratedModel, as required for computing partitions of finite ranges

- bool check_ranges_sets (int num_steps)

sanity check for vector parameter study

- bool check_ranges_sets (const IntVector &c_steps, const IntVector &di_steps, const IntVector &ds_steps, const IntVector &dr_steps)

sanity check for centered parameter study

- bool check_sets (const IntVector &c_steps, const IntVector &di_steps, const IntVector &ds_steps, const IntVector &dr_steps)

sanity check for increments along int/real set dimensions

- int integer_step (int range, int num_steps) const

check for integer remainder and return step

- int index_step (size_t start, size_t end, int num_steps) const

check for out of bounds and index remainder and return step

- void c_step (size_t c_index, int increment, Variables &vars)

helper function for performing a continuous step in one variable

- void dri_step (size_t di_index, int increment, Variables &vars)

helper function for performing a discrete step in an integer range variable

- void dsi_step (size_t di_index, int increment, const IntSet &values, Variables &vars)

helper function for performing a discrete step in an integer set variable

- void dss_step (size_t ds_index, int increment, const StringSet &values, Variables &vars)

helper function for performing a discrete step in a string set variable

- void dsr_step (size_t dr_index, int increment, const RealSet &values, Variables &vars)

helper function for performing a discrete step in a real set variable

- void reset (Variables &vars)

reset vars to initial point (center)

- void centered_header (const String &type, size_t var_index, int step, size_t hdr_index)

store a centered parameter study header within allHeaders
Private Attributes

- `size_t numEvals`
  total number of parameter study evaluations computed from specification
- `RealVectorArray listCVPoints`
  array of continuous evaluation points for the list parameter study
- `IntArrayArray listDIVPoints`
  array of discrete int evaluation points for the list parameter study
- `StringMulti2DArray listDSVPoints`
  array of discrete string evaluation points for the list parameter study
- `RealVectorArray listDRVPoints`
  array of discrete real evaluation points for the list parameter study
- `RealVector initialCVPoint`
  the continuous start point for vector and centered parameter studies
- `IntArray initialDIVPoint`
  the discrete int start point for vector and centered parameter studies
- `StringMultiArray initialDSVPoint`
  the discrete string start point for vector and centered parameter studies
- `RealVector initialDRVPoint`
  the discrete real start point for vector and centered parameter studies
- `RealVector finalCVPoint`
  the continuous ending point for vector parameter study
- `IntArray finalDIVPoint`
  the discrete int range value or set index ending point for vector parameter study
- `RealVector finalDRVPoint`
  the discrete real set index ending point for vector parameter study
- `RealVector contStepVector`
  the n-dimensional continuous increment
- `IntArray discIntStepVector`
  the n-dimensional discrete integer range value or set index increment
- `IntArray discStringStepVector`
  the n-dimensional discrete string set index increment
- `IntArray discRealStepVector`
  the n-dimensional discrete real set index increment
- `int numSteps`
  the number of times continuous/discrete step vectors are applied for vector parameter study (a specification option)
- `IntArray contStepsPerVariable`
  number of offsets in the plus and the minus direction for each continuous variable in a centered parameter study
- `IntArray discIntStepsPerVariable`
  number of offsets in the plus and the minus direction for each discrete integer variable in a centered parameter study
- `IntArray discStringStepsPerVariable`
  number of offsets in the plus and the minus direction for each discrete string variable in a centered parameter study
- `IntArray discRealStepsPerVariable`
Additional Inherited Members

13.149.1 Detailed Description

Class for vector, list, centered, and multidimensional parameter studies.

The `ParamStudy` class contains several algorithms for performing parameter studies of different types. The vector parameter study steps along an n-dimensional vector from an arbitrary initial point to an arbitrary final point in a specified number of steps. The centered parameter study performs a number of plus and minus offsets in each coordinate direction around a center point. A multidimensional parameter study fills an n-dimensional hypercube based on bounds and a specified number of partitions for each dimension. And the list parameter study provides for a user specification of a list of points to evaluate, which allows general parameter investigations not fitting the structure of vector, centered, or multidim parameter studies.

13.149.2 Member Function Documentation

```cpp
void pre_run() [virtual]
```

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `pre_run()`, if implemented, typically before performing its own implementation steps.

Reimplemented from Analyzer.

References Dakota::abort_handler(), SharedVariablesData::active_components_totals(), Analyzer::allHeaders, Analyzer::allVariables, ParamStudy::centered_loop(), Variables::continuous_variables(), ParamStudy::contStepsPerVariable, ParamStudy::contStepVector, ParamStudy::contVarPartitions, Dakota::copy_data(), Model::current_variables(), ParamStudy::discIntStepsPerVariable, ParamStudy::discIntStepVector, ParamStudy::discIntVarPartitions, ParamStudy::discRealStepsPerVariable, ParamStudy::discRealStepVector, ParamStudy::discRealVarPartitions, Variables::discrete_int_variables(), Variables::discrete_real_variables(), Variables::discrete_string_variables(), ParamStudy::discStringStepsPerVariable, ParamStudy::discStringStepVector, ParamStudy::discStringVarPartitions, ParamStudy::distribute_partitions(), ParamStudy::final_point_to_step_vector(), ParamStudy::finalCVPoint, ParamStudy::finalDIVPoint, ParamStudy::finalDRVPoint, ParamStudy::finalDSVPoint, ParamStudy::initialCVPoint, ParamStudy::initialDIVPoint, ParamStudy::initialDRVPoint, ParamStudy::initialDSVPoint, Iterator::iteratedModel, Iterator::method_enum_to_string(), Iterator::methodName, ParamStudy::multidim_loop(), Analyzer::numDiscreteStringVars, ParamStudy::numEvals, ParamStudy::numSteps, Iterator::outputLevel, Analyzer::pre_run(), ParamStudy::sample(), Variables::shared_data(), Dakota::svd(), ParamStudy::vector_loop(), and Dakota::write_ordered().
### void core_run() [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post steps in the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, Iterator::methodName, Analyzer::numLSqTerms, Analyzer::numObjFns, and Iterator::subIteratorFlag.

### void post_run(std::ostream & s) [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Analyzer::allResponses, Analyzer::allVariables, SensAnalysisGlobal::compute_correlations(), Model::discrete_set_string_values(), Iterator::iteratedModel, Iterator::methodName, Analyzer::post_run(), PStudyDAE::pStudyDACESensGlobal, and Iterator::subIteratorFlag.

### bool load_distribute_points(const String & points_filename, unsigned short tabular_format, bool active_only) [private]

load list of points from data file and distribute among listCVPoints, listDIVPoints, listDSVPoints, and listDRVPoints

Load from file and distribute points; using this function to manage construction of the temporary arrays. Historically all data was read as a real (mixture of values and indices), but now points_file is valued-based (reals, integers, strings) so file input matches tabular data output. Return false on success.

References Dakota::NPOS, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Variables::copy(), Model::current_variables(), Model::discrete_int_lower_bounds(), Model::discrete_int_sets(), Model::discrete_int_upper_bounds(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Iterator::iteratedModel, ParamStudy::listCVPoints, ParamStudy::listDIVPoints, ParamStudy::listDRVPoints, ParamStudy::listDSVPoints, ParamStudy::numContinuousVars, ParamStudy::numDiscreteIntVars, ParamStudy::numDiscreteRealVars, ParamStudy::numDiscreteStringVars, ParamStudy::numEvals, and Dakota::set_value_to_index().

Referenced by ParamStudy::ParamStudy().

### bool distribute_list_of_points(const RealVector & list_of_pts) [private]

distributes list_of_pts coming from user spec among listCVPoints, listDIVPoints, listDSVPoints, and listDRVPoints

Parse list of points into typed data containers; list_of_pts will contain values for continuous and discrete integer range, but indices for all discrete set types (int, string, real)

References Model::discrete_int_sets(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), ParamStudy::distribute(), Iterator::iteratedModel, ParamStudy::listCVPoints, ParamStudy::listDIVPoints, ParamStudy::listDSVPoints, ParamStudy::listDRVPoints, ParamStudy::numContinuousVars, ParamStudy::numDiscreteIntVars, ParamStudy::numDiscreteRealVars, ParamStudy::numDiscreteStringVars, ParamStudy::numEvals, and Dakota::set_index_to_value().

Referenced by ParamStudy::ParamStudy().

The documentation for this class was generated from the following files:
13.150 partial_prp_equality Struct Reference

predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

Public Member Functions

- bool operator() (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr) const

13.150.1 Detailed Description

predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

The documentation for this struct was generated from the following file:

- PRPMultiIndex.hpp

13.151 partial_prp_hash Struct Reference

wrapper to delegate to the ParamResponsePair hash_value function

Public Member Functions

- std::size_t operator() (const ParamResponsePair &prp) const

13.151.1 Detailed Description

wrapper to delegate to the ParamResponsePair hash_value function

The documentation for this struct was generated from the following file:

- PRPMultiIndex.hpp

13.152 PebbldBranching Class Reference

Main Branching class for the PEBBL-based Minimizer.
Inherits branching.

Public Member Functions

- PebbldBranching()
  Default Constructor.
- ~PebbldBranching()
  Destructor.
- pebbl::branchSub * blankSub()
Method that returns an empty Sub-Branch.

- void `setModel(Model &model)`
- void `setIterator(Iterator &iterator)`

**Protected Attributes**

- `Model parentModel`
  
  Original model, before branching.

- `Iterator nlpSolver`
  
  Solver to be used at root node.

- `RealVector cont_vars`
  
  Initial variable values for root node.

- `RealVector lower_bounds`
  
  Lower bounds for root node.

- `RealVector upper_bounds`
  
  Upper bounds for root node.

**Friends**

- class `PebbldBranchSub`

### 13.152.1 Detailed Description

Main Branching class for the PEBBL-based Minimizer.

The documentation for this class was generated from the following files:

- `PEBBLBranching.hpp`
- `PEBBLBranching.cpp`

### 13.153 PebbldBranchSub Class Reference

Sub Branch class for the PEBBL-based Minimizer.

Inherits `branchSub`.

**Public Member Functions**

- `PebbldBranchSub()`
  
  Constructor.

- `~PebbldBranchSub()`
  
  Destructor.

- `PebbldBranching * global() const`
  
  Returns a reference to the corresponding main Branching object.

- `pebl::branching * bGlobal() const`
  
  Returns a reference to the corresponding main Branching object.

- `void setGlobalInfo(PebbldBranching *global_)`
  
  Method that sets up the main Branching object.
• void setRootComputation ()
  Method that is called when declaring the current node as a root node.
• void boundComputation (double *controlParam)
  Method that does the Bounding Operation. In other words, it calls the optimization algorithm on the relaxed
  domain.
• bool candidateSolution ()
  Method called after the bounding operation, returns true if the bounding resulted in a possible solution to the
  original non-relaxed problem.
• pebbl::solution * extractSolution ()
  Method that returns a PEBBL-based solution.
• int splitComputation ()
  Method that determines how many branches are created and how. Returns the number of branches created from
  this object.
• pebbl::branchSub * makeChild (int whichChild)
  Method that returns a new PebbldBranchSub object based on Objective Function improvements and the number of
  branches.
• void pебbldSubAsChildOf (PebbldBranchSub *parent, int splitVar, int whichChild, std::vector< double >
  _candidate_x, RealVector _lower_bounds, RealVector _upper_bounds)
  Method that creates a new PebbldBranching object.

Protected Attributes

• PebbldBranching * globalPtr
  Pointer referring to all info passed to subproblem.
• Model subModel
  Model used for sub-problem.
• Iterator subNLPSolver
  Solver used for sub-problems.
• std::vector< double > candidate_x
  Candidate solution after solving sub-problem (also the bound)
• double candidate_objFn
  Objective value at the candidate solution.
• int splitVar
  Variable to branch on.
• RealVector cont_vars
  Initial variable values for sub-problem.
• RealVector lower_bounds
  Lower bounds for sub-problem.
• RealVector upper_bounds
  Upper bounds for sub-problem.

Friends

• class PebbldBranching
13.154. **PEBBLTRAITS CLASS REFERENCE**

### 13.153.1 Detailed Description

Sub Branch class for the PEBBL-based **Minimizer**.

The documentation for this class was generated from the following files:

- PEBBLBranching.hpp
- PEBBLBranching.cpp

### 13.154 PebbldTraits Class Reference

Wrapper class for experimental PebbldMinimizer.

Inheritance diagram for PebbldTraits:

```
TraitsBase

PebbldTraits
```

#### Public Member Functions

- **PebbldTraits ()**  
  *default constructor*
- virtual **~PebbldTraits ()**  
  *destructor*
- virtual bool **is_derived ()**  
  *A temporary query used in the refactor.*
- bool **supports_continuous_variables ()**  
  *Return the flag indicating whether method supports continuous variables.*
- bool **supports_discrete_variables ()**  
  *Return the flag indicating whether method supports discrete variables.*
- bool **supports_nonlinear_equality ()**  
  *Return the flag indicating whether method supports nonlinear equalities.*
- bool **supports_nonlinear_inequality ()**  
  *Return the flag indicating whether method supports nonlinear inequalities.*

### 13.154.1 Detailed Description

Wrapper class for experimental PebbldMinimizer.

A version of **TraitsBase** specialized for Pebbld optimizers

The documentation for this class was generated from the following file:

- PEBBLMinimizer.hpp
13.155 PecosApproximation Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for PecosApproximation:

```
Approximation
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
PecosApproximation
```

Public Member Functions

- **PecosApproximation ()**
  
  *default constructor*

- **PecosApproximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  *standard ProblemDescDB-driven constructor*

- **PecosApproximation (const SharedApproxData &shared_data)**
  
  *alternate constructor*

- **~PecosApproximation ()**
  
  *destructor*

- **void expansion_coefficient_flag (bool coeff_flag)**
  
  *set pecosBasisApprox.configOptions.expansionCoeffFlag*

- **bool expansion_coefficient_flag () const**
  
  *get pecosBasisApprox.configOptions.expansionCoeffFlag*

- **void expansion_gradient_flag (bool grad_flag)**
  
  *set pecosBasisApprox.configOptions.expansionGradFlag*

- **bool expansion_gradient_flag () const**
  
  *get pecosBasisApprox.configOptions.expansionGradFlag*

- **void compute_component_effects ()**
  
  *Performs global sensitivity analysis using Sobol' Indices by computing component (main and interaction) effects.*

- **void compute_total_effects ()**
  
  *Performs global sensitivity analysis using Sobol' Indices by computing total effects.*

- **const Pecos::RealVector & sobol_indices () const**
  
  *return polyApproxRep->sobolIndices*

- **const Pecos::RealVector & total_sobol_indices () const**
  
  *return polyApproxRep->totalSobolIndices*

- **Pecos::ULongULongMap sparse_sobol_index_map () const**
  
  *return RegressOrthogPolyApproximation::sparseSobolIndexMap*

- **const Pecos::RealVector & dimension_decay_rates () const**
  
  *return OrthogPolyApproximation::decayRates*

- **void allocate_arrays ()**
  
  *invoke Pecos::PolynomialApproximation::allocate_arrays()*
• Real mean ()
  return the mean of the expansion, treating all variables as random

• Real mean (const Pecos::RealVector &x)
  return the mean of the expansion for a given parameter vector, treating a subset of the variables as random

• const Pecos::RealVector & mean_gradient ()
  return the gradient of the expansion mean for a given parameter vector, treating all variables as random

• const Pecos::RealVector & mean_gradient (const Pecos::RealVector &, const Pecos::SizetArray &dvv)
  return the gradient of the expansion mean for a given parameter vector and given DVV, treating a subset of the variables as random

• Real variance ()
  return the variance of the expansion, treating all variables as random

• Real variance (const Pecos::RealVector &x)
  return the variance of the expansion for a given parameter vector, treating a subset of the variables as random

• const Pecos::RealVector & variance_gradient ()
  return the gradient of the expansion variance for a given parameter vector, treating all variables as random

• const Pecos::RealVector & variance_gradient (const Pecos::RealVector &, const Pecos::SizetArray &dvv)
  return the gradient of the expansion variance for a given parameter vector and given DVV, treating a subset of the variables as random

• Real covariance (PecosApproximation *pecos_approx_2)
  return the covariance between two response expansions, treating all variables as random

• Real covariance (const Pecos::RealVector &x, PecosApproximation *pecos_approx_2)
  return the covariance between two response expansions, treating a subset of the variables as random

• Real delta_covariance (PecosApproximation *pecos_approx_2)
  return the change in covariance between two response expansions, treating all variables as random

• Real delta_covariance (const Pecos::RealVector &x, PecosApproximation *pecos_approx_2)
  return the change in covariance between two response expansions, treating a subset of the variables as random

• Real delta_mean ()
  return the change in mean between two response expansions, treating all variables as random

• Real delta_mean (const RealVector &x)
  return the change in mean between two response expansions, treating a subset of variables as random

• Real delta_std_deviation ()
  return the change in standard deviation between two response expansions, treating all variables as random

• Real delta_std_deviation (const RealVector &x)
  return the change in standard deviation between two response expansions, treating a subset of variables as random

• Real delta_beta (bool cdf_flag, Real z_bar)
  return the change in reliability index (mapped from z_bar) between two response expansions, treating all variables as random

• Real delta_beta (const RealVector &, bool cdf_flag, Real z_bar)
  return the change in reliability index (mapped from z_bar) between two response expansions, treating a subset of variables as random

• Real delta_z (bool cdf_flag, Real beta_bar)
  return the change in response level (mapped from beta_bar) between two response expansions, treating all variables as random

• Real delta_z (const RealVector &, bool cdf_flag, Real beta_bar)
  return the change in response level (mapped from beta_bar) between two response expansions, treating a subset of variables as random
return the change in response level (mapped from beta_bar) between two response expansions, treating a subset of the variables as random

- **void compute_moments**(bool full_stats=true)
  compute moments up to the order supported by the Pecos polynomial approximation

- **void compute_moments**(const Pecos::RealVector &x, bool full_stats=true)
  compute moments in all-variables mode up to the order supported by the Pecos polynomial approximation

- **const RealVector & moments**() const
  return virtual Pecos::PolynomialApproximation::moments()

- **const RealVector & expansion_moments**() const
  return Pecos::PolynomialApproximation::expansionMoments

- **const RealVector & numerical_integration_moments**() const
  return Pecos::PolynomialApproximation::numericalMoments

- **void standardize_moments**(const Pecos::RealVector &central_moments, Pecos::RealVector &std_moments)
  standardize the central moments returned from Pecos

- **void build_linear_system**(RealMatrix &A, const UShort2DArray &multi_index)
  construct the Vandermonde matrix "A" for PCE regression for Ax = b

- **void augment_linear_system**(const RealVectorArray &samples, RealMatrix &A, const UShort2DArray &multi_index)

- **const Pecos::SurrogateData & surrogate_data**() const
  return Pecos::BasisApproximation & pecos_basis_approximation()

Protected Member Functions

- **Real value**(const Variables &vars)
  retrieve the approximate function value for a given parameter vector

- **const Pecos::RealVector & gradient**(const Variables &vars)
  retrieve the approximate function gradient for a given parameter vector

- **const Pecos::RealSymMatrix & hessian**(const Variables &vars)
  retrieve the approximate function Hessian for a given parameter vector

- **int min_coefficients**() const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-
  Vars dimensions

- **void build**(size_t index=_NPOS)
  builds the approximation from scratch

- **void rebuild**(size_t index=_NPOS)
  rebuilds the approximation incrementally

- **void pop**(bool save_data)
  removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)

- **void push**()
  restores state prior to previous pop()

- **void finalize**()
  finalize approximation by applying all remaining trial sets

- **void store**(size_t index=_NPOS)
13.155. PECOSAPPROXIMATION CLASS REFERENCE

store current approximation state for later combination
- void restore (size_t index=NPOS)
  restore previous approximation state
- void removeStored (size_t index=NPOS)
  remove a stored approximation prior to combination
- void combine (size_t swap_index)
  combine current approximation with previously stored approximation
- void clear_stored ()
  clear stored approximation data
- void print_coefficients (std::ostream &s, bool normalized)
  print the coefficient array computed in build()/rebuild()
- RealVector approximation_coefficients (bool normalized) const
  return expansion coefficients in a form consistent with the shared multi-index
- void approximation_coefficients (const RealVector &approx_coeffs, bool normalized)
  set expansion coefficients in a form consistent with the shared multi-index
- void coefficient_labels (std::vector<std::string> &coeff_labels) const
  print the coefficient array computed in build()/rebuild()

Private Member Functions
- void approx_type_to_basis_type (const String &approx_type, short &basis_type)
  utility to convert Dakota type string to Pecos type enumeration

Private Attributes
- Pecos::BasisApproximation pecosBasisApprox
  the Pecos basis approximation, encompassing OrthogPolyApproximation and InterpPolyApproximation
- Pecos::PolynomialApproximation * polyApproxRep
  convenience pointer to representation of Pecos polynomial approximation

Additional Inherited Members

13.155.1 Detailed Description

Derived approximation class for global basis polynomials.

The PecosApproximation class provides a global approximation based on basis polynomials. This includes orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic collocation.

13.155.2 Member Function Documentation

void build ( size_t index = NPOS ) [inline], [protected], [virtual]

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.

References Approximation::build(), and PecosApproximation::pecosBasisApprox.
void rebuild ( size_t index = _NPOS ) [inline], [protected], [virtual]

rebuilds the approximation incrementally

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations
should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References PecosApproximation::pecosBasisApprox.

void pop ( bool save data ) [inline], [protected], [virtual]

removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations
should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References PecosApproximation::pecosBasisApprox, and Approximation::pop().

void push ( ) [inline], [protected], [virtual]

restores state prior to previous pop()

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations
should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References PecosApproximation::pecosBasisApprox, and Approximation::push().

void finalize ( ) [inline], [protected], [virtual]

finalize approximation by applying all remaining trial sets

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations
should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.
References Approximation::finalize(), and PecosApproximation::pecosBasisApprox.

The documentation for this class was generated from the following files:

- PecosApproximation.hpp
- PecosApproximation.cpp

13.156 ProbabilityTransformModel Class Reference

Probability transformation specialization of RecastModel.

Inheritance diagram for ProbabilityTransformModel:

```
Model
|
| RecastModel
|
| ProbabilityTransformModel
```
Public Member Functions

- **ProbabilityTransformModel** (const Model &sub_model, bool truncated_bounding=false, Real bound=10.)  
  standard constructor
- ~**ProbabilityTransformModel** ()  
  destructor
- bool **initialize_mapping** (ParLevLIter p_iter)  
  initialize model mapping, returns true if the variables size has changed
- bool **finalize_mapping** ()  
  finalize model mapping, returns true if the variables size has changed
- bool **mapping_initialized** () const  
  return true if mapping has been fully initialized, false otherwise.
- bool **resize_pending** () const  
  return true if a potential resize is still pending, such that sizing-based initialization should be deferred
- void **transform_correlations** ()  
  perform correlation warping for variable types supported by Nataf

Protected Member Functions

- void **distribution_parameter_derivatives** (bool dist_param_derivs)  
  set distParamDerivs
- void **initialize_random_variables** (const Pecos::ProbabilityTransformation &transform, bool deep_copy=false)  
  alternate form: initialize natafTransform based on incoming data
- void **initialize_random_variable_transformation** ()  
  instantiate natafTransform
- void **initialize_random_variable_types** ()  
  initializes ranVarTypesX within natafTransform (u-space not needed)
- void **initialize_random_variable_types** (short u_space_type)  
  initializes ranVarTypesX and ranVarTypesU within natafTransform
- void **initialize_random_variable_parameters** ()  
  initializes ranVarMeansX, ranVarStdDevsX, ranVarLowerBndsX, ranVarUpperBndsX, and ranVarAddtlParamsX  
  within natafTransform
- void **initialize_random_variable_correlations** ()  
  propagate iteratedModel correlations to natafTransform
- void **verify_correlation_support** (short u_space_type)  
  verify that correlation warping is supported by Nataf for given variable types
- void **transform_model** (bool truncated_bound, Real bound)  
  recast x_model from x-space to u-space to create u_model
- unsigned short **pecos_to_dakota_variable_type** (unsigned short pecos_var_type)  
  convert from Pecos To Dakota variable enumeration type for continuous aleatory uncertain variables used in variable transformations
- void **initialize_sizes** ()
CHAPTER 13. CLASS DOCUMENTATION

Static Protected Member Functions

- static void vars_u_to_x_mapping (const Variables &u_vars, Variables &x_vars)
  static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations
- static void vars_x_to_u_mapping (const Variables &x_vars, Variables &u_vars)
  static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for NonD Iterators
- static void set_u_to_x_mapping (const Variables &u_vars, const ActiveSet &u_set, ActiveSet &x_set)
  static function for RecastModels used to map u-space ActiveSets from NonD Iterators to x-space ActiveSets for Model evaluations
- static void resp_x_to_u_mapping (const Variables &x_vars, const Variables &u_vars, const Response &x_response, Response &u_response)
  static function for RecastModels used to map x-space responses from Model evaluations to u-space responses for return to NonD Iterator.

Private Attributes

- Pecos::ProbabilityTransformation natafTransform
  Nonlinear variable transformation that encapsulates the required data for performing transformations from X -> Z -> U and back.
- size_t numFunctions
  number of response functions
- size_t numContinuousVars
  number of active continuous vars
- size_t numDiscreteIntVars
  number of active discrete integer vars
- size_t numDiscreteStringVars
  number of active discrete string vars
- size_t numDiscreteRealVars
  number of active discrete real vars
- size_t numContDesVars
  number of continuous design variables (modeled using uniform distribution for All view modes)
- size_t numDiscIntDesVars
  number of discrete integer design variables (modeled using discrete histogram distributions for All view modes)
- size_t numDiscStringDesVars
  number of discrete string design variables (modeled using discrete histogram distributions for All view modes)
- size_t numDiscRealDesVars
  number of discrete real design variables (modeled using discrete histogram distributions for All view modes)
- size_t numDesignVars
  total number of design variables
- size_t numContStateVars
  number of continuous state variables (modeled using uniform distribution for All view modes)
- size_t numDiscIntStateVars
  number of discrete integer state variables (modeled using discrete histogram distributions for All view modes)
• `size_t numDiscStringStateVars`
  number of discrete string state variables (modeled using discrete histogram distributions for All view modes)

• `size_t numDiscRealStateVars`
  number of discrete real state variables (modeled using discrete histogram distributions for All view modes)

• `size_t numStateVars`
  total number of state variables

• `size_t numNormalVars`
  number of normal uncertain variables (native space)

• `size_t numLognormalVars`
  number of lognormal uncertain variables (native space)

• `size_t numUniformVars`
  number of uniform uncertain variables (native space)

• `size_t numLoguniformVars`
  number of loguniform uncertain variables (native space)

• `size_t numTriangularVars`
  number of triangular uncertain variables (native space)

• `size_t numExponentialVars`
  number of exponential uncertain variables (native space)

• `size_t numBetaVars`
  number of beta uncertain variables (native space)

• `size_t numGammaVars`
  number of gamma uncertain variables (native space)

• `size_t numGumbelVars`
  number of gumbel uncertain variables (native space)

• `size_t numFrechetVars`
  number of frechet uncertain variables (native space)

• `size_t numWeibullVars`
  number of weibull uncertain variables (native space)

• `size_t numHistogramBinVars`
  number of histogram bin uncertain variables (native space)

• `size_t numPoissonVars`
  number of Poisson uncertain variables (native space)

• `size_t numBinomialVars`
  number of binomial uncertain variables (native space)

• `size_t numNegBinomialVars`
  number of negative binomial uncertain variables (native space)

• `size_t numGeometricVars`
  number of geometric uncertain variables (native space)

• `size_t numHyperGeomVars`
  number of hypergeometric uncertain variables (native space)

• `size_t numHistogramPtIntVars`
  number of histogram point integer uncertain variables (native space)

• `size_t numHistogramPtStringVars`
CHAPTER 13. CLASS DOCUMENTATION

- number of histogram point string uncertain variables (native space)
  - size_t numHistagramPtRealVars
- number of histogram point real uncertain variables (native space)
  - size_t numContIntervalVars
- number of continuous interval uncertain variables (native space)
  - size_t numDiscIntervalVars
- number of discrete interval uncertain variables (native space)
  - size_t numDiscSetIntUncVars
- number of discrete integer set uncertain variables (native space)
  - size_t numDiscSetStringUncVars
- number of discrete string set uncertain variables (native space)
  - size_t numDiscSetRealUncVars
- number of discrete real set uncertain variables (native space)
  - size_t numContAleatUncVars
- total number of continuous aleatory uncertain variables (native space)
  - size_t numDiscIntAleatUncVars
- total number of discrete integer aleatory uncertain variables (native space)
  - size_t numDiscStringAleatUncVars
- total number of discrete string aleatory uncertain variables (native space)
  - size_t numDiscRealAleatUncVars
- total number of discrete real aleatory uncertain variables (native space)
  - size_t numAleatoryUncVars
- total number of aleatory uncertain variables (native space)
  - size_t numContEpistUncVars
- total number of continuous epistemic uncertain variables (native space)
  - size_t numDiscIntEpistUncVars
- total number of discrete integer epistemic uncertain variables (native space)
  - size_t numDiscStringEpistUncVars
- total number of discrete string epistemic uncertain variables (native space)
  - size_t numDiscRealEpistUncVars
- total number of discrete real epistemic uncertain variables (native space)
  - size_t numEpistemicUncVars
- total number of epistemic uncertain variables (native space)
  - size_t numUncertainVars
- total number of uncertain variables (native space)
  - bool distParamDerivs
    - flags calculation of derivatives with respect to distribution parameters s within resp_x_to_u_mapping() using the
      chain rule df/dx dx/ds. The default is to calculate derivatives with respect to standard random variables u using the
      chain rule df/dx dx/du.
  - bool truncatedBounds
    - boolean flag to indicate truncated bounds
  - Real boundVal
    - bound value
13.156. PROBABILITYTRANSFORMMODEL CLASS REFERENCE

- **bool epistemicStats**
  
  flag for computing interval-type metrics instead of integrated metrics. If any epistemic variables are active in a metric evaluation, then this flag is set.

- **SizeArray primaryACVarMapIndices**
  
  "primary" all continuous variable mapping indices flowed down from higher level iteration

- **ShortArray secondaryACVarMapTargets**
  
  "secondary" all continuous variable mapping targets flowed down from higher level iteration

- **bool mappingInitialized**
  
  track use of initialize_mapping() and finalize_mapping()

**Static Private Attributes**

- **static ProbabilityTransformModel * ptmInstance**
  
  static pointer to this class for use in static callbacks

**Additional Inherited Members**

13.156.1 Detailed Description

Probability transformation specialization of RecastModel. Specialization of RecastModel to transform a sub-model to u-space.

13.156.2 Member Function Documentation

**void initialize_random_variables ( const Pecos::ProbabilityTransformation & transform, bool deep_copy = false ) [protected]**

alternate form: initialize natafTransform based on incoming data

This function is commonly used to publish transformation data when the Model variables are in a transformed space (e.g., u-space) and ProbabilityTransformation::ranVarTypes et al. may not be generated directly. This allows for the use of inverse transformations to return the transformed space variables to their original states.

References ProbabilityTransformModel::initialize_random_variable_transformation(), ProbabilityTransformModel::natafTransform, ProbabilityTransformModel::numContDesVars, ProbabilityTransformModel::numContEpistUncVars, ProbabilityTransformModel::numContIntervalVars, and ProbabilityTransformModel::numContStateVars.

**void initialize_random_variable_types ( ) [protected]**

initializes ranVarTypesX within natafTransform (u-space not needed)

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

References Model::aleatory_distribution_parameters(), Model::cv(), ProbabilityTransformModel::natafTransform, ProbabilityTransformModel::numBetaVars, ProbabilityTransformModel::numBinomialVars, ProbabilityTransformModel::numContDesVars, ProbabilityTransformModel::numContIntervalVars, ProbabilityTransformModel::numContStateVars, ProbabilityTransformModel::numExponentialVars, ProbabilityTransformModel::numFrechetVars,
CHAPTER 13. CLASS DOCUMENTATION


Referenced by ProbabilityTransformModel::ProbabilityTransformModel().

void initialize_random_variable_types ( short u_space_type ) [protected]

initializes ranVarTypesX and ranVarTypesU within natafTransform

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.


Referenced by ProbabilityTransformModel::initialize_mapping().

void initialize_random_variable_parameters ( ) [protected]

initializes ranVarMeansX, ranVarStdDevsX, ranVarLowerBndsX, ranVarUpperBndsX, and ranVarAddtlParamsX within natafTransform

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.


Referenced by ProbabilityTransformModel::initialize_mapping().

void vars_u_to_x_mapping ( const Variables & u_vars, Variables & x_vars ) [inline], [static], [protected]

static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations

Map the variables from iterator space (u) to simulation space (x).

References Variables::continuous_variables(), Variables::continuous_variables_view(), ProbabilityTransformModel::natafTransform, and ProbabilityTransformModel::ptmInstance.

Referenced by ProbabilityTransformModel::ProbabilityTransformModel().
**void vars_x_to_u_mapping ( const Variables & x_vars, Variables & u_vars ) [inline], [static], [protected]**

static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for **NonD** Iterators

- Map the variables from simulation space (x) to iterator space (u).
- References Variables::continuous_variables(), Variables::continuous_variables_view(), ProbabilityTransformModel::natafTransform, and ProbabilityTransformModel::ptmInstance.
- Referenced by ProbabilityTransformModel::ProbabilityTransformModel().

**void set_u_to_x_mapping ( const Variables & u_vars, const ActiveSet & u_set, ActiveSet & x_set ) [static], [protected]**

static function for RecastModels used to map u-space ActiveSets from **NonD** Iterators to x-space ActiveSets for **Model** evaluations

- Define the DVV for x-space derivative evaluations by augmenting the iterator requests to account for correlations.
- References Dakota::NPOS, Variables::all_continuous_variable_ids(), Dakota::contains(), Variables::continuous_variable_ids(), ActiveSet::derivative_vector(), Dakota::find_index(), Variables::inactive_continuous_variable_ids(), ProbabilityTransformModel::natafTransform, and ProbabilityTransformModel::ptmInstance.
- Referenced by ProbabilityTransformModel::ProbabilityTransformModel().

### 13.156.3 Member Data Documentation

**ProbabilityTransformModel * ptmInstance [static], [private]**

static pointer to this class for use in static callbacks

- initialization of static needed by **RecastModel**

Referenced by ProbabilityTransformModel::ProbabilityTransformModel(), ProbabilityTransformModel::resp_x_to_u_mapping(), ProbabilityTransformModel::set_u_to_x_mapping(), ProbabilityTransformModel::vars_u_to_x_mapping(), and ProbabilityTransformModel::vars_x_to_u_mapping().

The documentation for this class was generated from the following files:

- ProbabilityTransformModel.hpp
- ProbabilityTransformModel.cpp

### 13.157 ProblemDescDB Class Reference

The database containing information parsed from the DAKOTA input file.

Inheritance diagram for ProblemDescDB:

```
ProblemDescDB
  |__ NIDRProblemDescDB
```
Public Member Functions

- **ProblemDescDB ()**
  
  *default constructor*

- **ProblemDescDB (ParallelLibrary &parallel_lib)**
  
  *standard constructor*

- **ProblemDescDB (const ProblemDescDB &db)**
  
  *copy constructor*

- **~ProblemDescDB ()**
  
  *destructor*

- **ProblemDescDB operator= (const ProblemDescDB &db)**
  
  *assignment operator*

- **void parse_inputs (const ProgramOptions &prog_opts, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)**
  
  Parses the input file or input string if present and executes callbacks. Does not perform any validation.

- **void check_and_broadcast (const ProgramOptions &prog_opts)**
  
  Performs check_input, broadcast, and post_process, but for now, allowing separate invocation through the public API as well.

- **void check_input ()**
  
  Verifies that there is at least one of each of the required keywords in the dakota input file.

- **void broadcast ()**
  
  Invokes send_db_buffer() and receive_db_buffer() to broadcast DB data across the processor allocation. Used by manage_inputs().

- **void post_process ()**
  
  Post-processes the (minimal) input specification to assign default variables/responses specification arrays. Used by manage_inputs().

- **void lock ()**
  
  Locks the database in order to prevent data access when the list nodes may not be set properly. Unlocked by a set nodes operation.

- **void unlock ()**
  
  Explicitly unlocks the database. Use with care.

- **void set_db_list_nodes (const String &method_tag)**
  
  Set dataMethodIter based on a method identifier string to activate a particular method specification in dataMethodList and use pointers from this method specification to set all other list iterators.

- **void set_db_list_nodes (size_t method_index)**
  
  Set dataMethodIter based on an index within dataMethodList to activate a particular method specification and use pointers from this method specification to set all other list iterators.

- **void resolve_top_method (bool set_model_nodes=true)**
  
  For a (default) environment lacking a top method pointer, this function is used to determine which of several potential method specifications corresponds to the top method and then sets the list nodes accordingly.

- **void set_db_method_node (const String &method_tag)**
  
  Set dataMethodIter based on a method identifier string to activate a particular method specification (only).

- **void set_db_method_node (size_t method_index)**
  
  Set dataMethodIter based on an index within dataMethodList to activate a particular method specification (only).

- **size_t get_db_method_node ()**
return the index of the active node in dataMethodList

- void set_db_model_nodes (const String &model_tag)
  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on the model identifier string

- void set_db_model_nodes (size_t model_index)
  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on an index within dataModelList

- size_t get_db_model_node ()
  return the index of the active node in dataModelList

- void set_db_variables_node (const String &variables_tag)
  set dataVariablesIter based on the variables identifier string

- void set_db_interface_node (const String &interface_tag)
  set dataInterfaceIter based on the interface identifier string

- void set_db_responses_node (const String &responses_tag)
  set dataResponsesIter based on the responses identifier string

- ParallelLibrary & parallel_library () const
  return the parallelLib reference

- IteratorList & iterator_list ()
  return a list of all Iterator objects that have been instantiated

- ModelList & model_list ()
  return a list of all Model objects that have been instantiated

- VariablesList & variables_list ()
  return a list of all Variables objects that have been instantiated

- InterfaceList & interface_list ()
  return a list of all Interface objects that have been instantiated

- ResponseList & response_list ()
  return a list of all Response objects that have been instantiated

- const RealMatrixArray & get_rma (const String &entry_name) const
  get a RealMatrixArray out of the database based on an identifier string

- const RealVector & get_rv (const String &entry_name) const
  get a RealVector out of the database based on an identifier string

- const IntVector & get_iv (const String &entry_name) const
  get an IntVector out of the database based on an identifier string

- const BitArray & get_ba (const String &entry_name) const
  get a BitArray out of the database based on an identifier string

- const SizetArray & get_sza (const String &entry_name) const
  get a SizetArray out of the database based on an identifier string

- const UShortArray & get_usa (const String &entry_name) const
  get an UShortArray out of the database based on an identifier string

- const RealSymMatrix & get_rsm (const String &entry_name) const
  get a RealSymMatrix out of the database based on an identifier string

- const RealVectorArray & get_rva (const String &entry_name) const
  get a RealVectorArray out of the database based on an identifier string
• const IntVectorArray & get_iva (const String &entry_name) const
  get an IntVectorArray out of the database based on an identifier string
• const IntSet & get_is (const String &entry_name) const
  get an IntSet out of the database based on an identifier string
• const IntSetArray & get_isa (const String &entry_name) const
  get an IntSetArray out of the database based on an identifier string
• const StringSetArray & get_ssa (const String &entry_name) const
  get a StringSetArray out of the database based on an identifier string
• const RealSetArray & get_rsa (const String &entry_name) const
  get a RealSetArray out of the database based on an identifier string
• const IntRealMapArray & get_irma (const String &entry_name) const
  get an IntRealMapArray out of the database based on an identifier string
• const StringRealMapArray & get_srma (const String &entry_name) const
  get a StringRealMapArray out of the database based on an identifier string
• const RealRealMapArray & get_rrma (const String &entry_name) const
  get a RealRealMapArray out of the database based on an identifier string
• const RealRealPairRealMapArray & get_rrrma (const String &entry_name) const
  get a RealRealPairRealMapArray out of the database based on an identifier string
• const IntIntPairRealMapArray & get_iirma (const String &entry_name) const
  get an IntIntPairRealMapArray out of the database based on an identifier string
• const StringArray & get_sa (const String &entry_name) const
  get a StringArray out of the database based on an identifier string
• const String2DArray & get_s2a (const String &entry_name) const
  get a String2DArray out of the database based on an identifier string
• const String & get_string (const String &entry_name) const
  get a String out of the database based on an identifier string
• const Real & get_real (const String &entry_name) const
  get a Real out of the database based on an identifier string
• int get_int (const String &entry_name) const
  get an int out of the database based on an identifier string
• short get_short (const String &entry_name) const
  get a short out of the database based on an identifier string
• unsigned short get_ushort (const String &entry_name) const
  get an unsigned short out of the database based on an identifier string
• size_t get_sizet (const String &entry_name) const
  get a size_t out of the database based on an identifier string
• bool get_bool (const String &entry_name) const
  get a bool out of the database based on an identifier string
• void ** get_voidss (const String &entry_name) const
  for getting a void**, e.g., &dlLib
• void insert_node (const DataEnvironment &data_env)
  set the DataEnvironment object
• void insert_node (const DataMethod &data_method)
add a DataMethod object to the dataMethodList

- void insert_node (const DataModel &data_model)
  add a DataModel object to the dataModelList
- void insert_node (DataVariables &data_variables)
  add a DataVariables object to the dataVariablesList
- void insert_node (const DataInterface &data_interface)
  add a DataInterface object to the dataInterfaceList
- void insert_node (const DataResponses &data_responses)
  add a DataResponses object to the dataResponsesList

- void set (const String &entry_name, const RealVector &rv)
  set a RealVector within the database based on an identifier string
- void set (const String &entry_name, const IntVector &iv)
  set an IntVector within the database based on an identifier string
- void set (const String &entry_name, const BitArray &ba)
  set a BitArray within the database based on an identifier string
- void set (const String &entry_name, const RealSymMatrix &rsm)
  set a RealSymMatrix within the database based on an identifier string
- void set (const String &entry_name, const RealVectorArray &rva)
  set a RealVectorArray within the database based on an identifier string
- void set (const String &entry_name, const IntVectorArray &iva)
  set an IntVectorArray within the database based on an identifier string
- void set (const String &entry_name, const IntSetArray &isa)
  set an IntSetArray within the database based on an identifier string
- void set (const String &entry_name, const RealSetArray &rsa)
  set a RealSetArray within the database based on an identifier string
- void set (const String &entry_name, const IntRealMapArray &irma)
  set an IntRealMapArray within the database based on an identifier string
- void set (const String &entry_name, const StringRealMapArray &srma)
  set a StringRealMapArray within the database based on an identifier string
- void set (const String &entry_name, const RealRealMapArray &rrma)
  set a RealRealMapArray within the database based on an identifier string
- void set (const String &entry_name, const RealRealPairRealMapArray &iirma)
  set a RealRealPairRealMapArray in the db based on an identifier string
- void set (const String &entry_name, const IntIntPairRealMapArray &iirma)
  set an IntIntPairRealMapArray in the db based on an identifier string
- void set (const String &entry_name, const StringArray &sa)
  set a StringArray within the database based on an identifier string

- int min_procs_per_ea ()
  compute minimum evaluation partition size based on lower level overrides
- int max_procs_per_ea ()
  compute maximum evaluation partition size based on lower level overrides and concurrency levels
- int min_procs_per_ie ()
  compute minimum iterator partition size based on lower level overrides
• int max_procs_per_ie (int max_eval_concurrency)
  compute maximum iterator partition size based on lower level overrides and concurrency levels
• bool method_locked () const
  return methodDBLocked
• bool model_locked () const
  return modelDBLocked
• bool variables_locked () const
  return variablesDBLocked
• bool interface_locked () const
  return interfaceDBLocked
• bool responses_locked () const
  return responsesDBLocked
• bool is_null () const
  function to check dbRep (does this envelope contain a letter)

**Static Public Member Functions**

• static int min_procs_per_level (int min_procs_per_server, int pps_spec, int num_serv_spec)
  compute minimum partition size for a parallel level based on lower level overrides
• static int max_procs_per_level (int max_procs_per_server, int pps_spec, int num_serv_spec, short sched_spec, int async_local_conc, bool peer_dynamic_avail, int max_concurrency)
  compute maximum partition size for a parallel level based on lower level overrides

**Protected Member Functions**

• ProblemDescDB (BaseConstructor, ParallelLibrary &parallel_lib)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
• virtual void derived_parse_inputs (const ProgramOptions &prog_opts)
  derived class specifics within parse_inputs()
• virtual void derived_broadcast ()
  derived class specifics within broadcast()
• virtual void derived_post_process ()
  derived class specifics within post_process()

**Protected Attributes**

• DataEnvironment environmentSpec
  the environment specification (only one allowed) resulting from a call to environment_kwhandler() or insert_node()
• std::list<DataMethod> dataMethodList
  list of method specifications, one for each call to method_kwhandler() or insert_node()
• std::list<DataModel> dataModelList
  list of model specifications, one for each call to model_kwhandler() or insert_node()
• std::list<DataVariables> dataVariablesList
  list of variables specifications, one for each call to variables_kwhandler() or insert_node()
13.157. PROBLEMDESCDB CLASS REFERENCE

- std::list< DataInterface > dataInterfaceList
  list of interface specifications, one for each call to interface_kwhandler() or insert_node()
- std::list< DataResponses > dataResponsesList
  list of responses specifications, one for each call to responses_kwhandler() or insert_node()
- size_t environmentCntr
  counter for environment specifications used in check_input

Private Member Functions

- const Iterator & get_iterator ()
  retrieve an existing Iterator, if it exists in iteratorList, or instantiate a new one
- const Iterator & get_iterator (Model &model)
  retrieve an existing Iterator, if it exists in iteratorList, or instantiate a new one
- const Iterator & get_iterator (const String &method_name, Model &model)
  retrieve an existing Iterator, if it exists in iteratorByNameList, or instantiate a new one
- const Model & get_model ()
  retrieve an existing Model, if it exists, or instantiate a new one
- const Variables & get_variables ()
  retrieve an existing Variables, if it exists, or instantiate a new one
- const Interface & get_interface ()
  retrieve an existing Interface, if it exists, or instantiate a new one
- const Response & get_response (short type, const Variables &vars)
  retrieve an existing Response, if it exists, or instantiate a new one
- ProblemDescDB * get_db (ParallelLibrary &parallel_lib)
  Used by the envelope constructor to instantiate the correct letter class.
- void send_db_buffer ()
  MPI send of a large buffer containing environmentSpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage_inputs().
- void receive_db_buffer ()
  MPI receive of a large buffer containing environmentSpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage_inputs().
- bool model_has_interface (DataModelRep *model_rep) const
  helper function for determining whether an interface specification should be active, based on model type
- void echo_input_file (const ProgramOptions &prog_opts)
  echo the (potentially) specified input file or string to stdout

Private Attributes

- ParallelLibrary & parallelLib
  reference to the parallel_lib object passed from main
- std::list< DataMethod >::iterator dataMethodIter
  iterator identifying the active list node in dataMethodList
- std::list< DataModel >::iterator dataModelIter
  iterator identifying the active list node in dataModelList
• std::list< DataVariables >
  ::iterator dataVariablesIter
   iterator identifying the active list node in dataVariablesList

• std::list< DataInterface >
  ::iterator dataInterfaceIter
   iterator identifying the active list node in dataInterfaceList

• std::list< DataResponses >
  ::iterator dataResponsesIter
   iterator identifying the active list node in dataResponsesList

• IteratorList iteratorList
   list of iterator objects, one for each method specification

• IteratorList iteratorByNameList
   list of iterator objects, one for each lightweight instantiation by name

• ModelList modelList
   list of model objects, one for each model specification

• VariablesList variablesList
   list of variables objects, one for each variables specification

• InterfaceList interfaceList
   list of interface objects, one for each interface specification

• ResponseList responseList
   list of response objects, one for each responses specification

• bool methodDBLocked
   prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active method specification

• bool modelDBLocked
   prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active model specification

• bool variablesDBLocked
   prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active variables specification

• bool interfaceDBLocked
   prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active interface specification

• bool responsesDBLocked
   prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active responses specification

• ProblemDescDB * dbRep
   pointer to the letter (initialized only for the envelope)

• int referenceCount
   number of objects sharing dbRep
13.157. PROBLEMDESCDB CLASS REFERENCE

Friends

- class Model
  
  Model requires access to `get_variables()` and `get_response()`

- class SimulationModel
  
  SimulationModel requires access to `get_interface()`

- class HierarchSurrModel
  
  HierarchSurrModel requires access to `get_model()`

- class DataFitSurrModel
  
  DataFitSurrModel requires access to `get_iterator()` and `get_model()`

- class NestedModel
  
  NestedModel requires access to `get_interface()`, `get_response()`, `get_iterator()`, and `get_model()`

- class ActiveSubspaceModel

- class AdaptedBasisModel

- class RandomFieldModel

- class Environment
  
  Environment requires access to `get_iterator()`

- class IteratorScheduler
  
  Environment requires access to `get_iterator()`

- class Iterator
  
  Iterator requires access to `get_model()`

- class MetaIterator
  
  MetaIterator requires access to `get_model()`

- class SeqHybridMetaIterator
  
  SeqHybridMetaIterator requires access to `get_model()`

- class CollabHybridMetaIterator
  
  CollabHybridMetaIterator requires access to `get_model()`

- class ConcurrentMetaIterator
  
  ConcurrentMetaIterator requires access to `get_model()`

- class SurrBasedLocalMinimizer
  
  HierarchSurrBasedLocalMinimizer requires access to `get_iterator()`

- class SurrBasedGlobalMinimizer
  
  SurrBasedGlobalMinimizer requires access to `get_iterator()`

- class PebbldMinimizer
  
  PEBBLMinimizer requires access to `get_iterator()`

13.157.1 Detailed Description

The database containing information parsed from the DAKOTA input file.

The ProblemDescDB class is a database for DAKOTA input file data that is populated by a parser defined in a derived class. When the parser reads a complete keyword, it populates a data class object (DataEnvironment, DataMethod, DataVariables, DataInterface, or DataResponses) and, for all cases except environment, appends the object to a linked list (dataMethodList, dataVariablesList, dataInterfaceList, or dataResponsesList). No environment linked list is used since only one environment specification is allowed.
13.157.2 Constructor & Destructor Documentation

**ProblemDescDB ( )**

Default constructor

The default constructor: dbRep is NULL in this case. This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

**ProblemDescDB ( ParallelLibrary & parallel_lib )**

Standard constructor

This is the envelope constructor which uses problem_db to build a fully populated db object. It only needs to extract enough data to properly execute get_db(problem_db), since the constructor overloaded with Base-Constructor builds the actual base class data inherited by the derived classes.

References Dakota::abort_handler(), ProblemDescDB::dbRep, and ProblemDescDB::get_db().

**ProblemDescDB ( const ProblemDescDB & db )**

Copy constructor

Copy constructor manages sharing of dbRep and incrementing of referenceCount.

References ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

**ProblemDescDB ( BaseConstructor , ParallelLibrary & parallel_lib ) [protected]**

Constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_db() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_db() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~ProblemDescDB).

13.157.3 Member Function Documentation

**ProblemDescDB operator= ( const ProblemDescDB & db )**

Assignment operator


References ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

**void parse_inputs ( const ProgramOptions & prog_opts, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL )**

Parses the input file or input string if present and executes callbacks. Does not perform any validation.

DB setup phase 1: parse the input file and execute callback functions if present. Rank 0 only.

DB setup phase 2: optionally insert additional data via late sets. Rank 0 only.
References Dakota::abort_handler(), ProblemDescDB::dbRep, ProblemDescDB::derived_parse_inputs(), Program-
Options::echo_input(), ProblemDescDB::echo_input_file(), ProgramOptions::input_file(), ProgramOptions::input-
_string(), ProblemDescDB::parallelLib, ProblemDescDB::parse_inputs(), and ParallelLibrary::world_rank().
Referenced by Environment::parse(), and ProblemDescDB::parse_inputs().

```cpp
void check_and_broadca(d ( const ProgramOptions & prog_opts )
```

performs check_input, broadcast, and post_process, but for now, allowing separate invocation through the public
API as well

DB setup phase 3: perform basic checks on keywords counts in current DB state, then sync to all processors.
References ProblemDescDB::broadcast(), ProblemDescDB::check_and_broadcast(), ProblemDescDB::check-
input(), ProblemDescDB::dbRep, ProblemDescDB::parallelLib, ProblemDescDB::post_process(), and Parallel-
Library::world_rank().
Referenced by ProblemDescDB::check_and_broadcast(), LibraryEnvironment::done_modifying_db(), and Environment-
::parse().

```cpp
void check_input ( )
```

verifies that there is at least one of each of the required keywords in the dakota input file

NOTE: when using library mode in a parallel application, check_input() should either be called only on world-
Rank 0, or it should follow a matched send_db_buffer() / receive_db_buffer() pair.
References Dakota::abort_handler(), ParallelLibrary::command_line_post-
run_input(), ParallelLibrary::command_line_post_run_output(), ParallelLibrary::command_line_pre_run_input(), Parallel-
Library::command_line_pre_run_output(), ParallelLibrary::command_line_run_input(), ParallelLibrary::command-
_line_run_output(), ParallelLibrary::command_line_user_modes(), ProblemDescDB::dataInterfaceList, ProblemDesc-
DB::dataMethodList, ProblemDescDB::dataModelList, ProblemDescDB::dataResponsesList, ProblemDescDB-
::dataVariablesList, ProblemDescDB::dbRep, ProblemDescDB::environmentCntr, ProblemDescDB::parallelLib, and Dakota::strbegins().
Referenced by ProblemDescDB::check_and_broadcast(), and ProblemDescDB::check_input().

```cpp
void post_process ( )
```

post-processes the (minimal) input specification to assign default variables/responses specification arrays. Used
by manage_inputs().

When using library mode in a parallel application, post_process() should be called on all processors following
broadcast() of a minimal problem specification.
References ProblemDescDB::dbRep, and ProblemDescDB::derived_post_process().
Referenced by ProblemDescDB::check_and_broadcast().

```cpp
ProblemDescDB * get_db ( ParallelLibrary & parallel_lib ) [private]
```

Used by the envelope constructor to instantiate the correct letter class.

Initializes dbRep to the appropriate derived type. The standard derived class constructors are invoked.
References Dakota::Dak_pddb.
Referenced by ProblemDescDB::ProblemDescDB().
The documentation for this class was generated from the following files:

- ProblemDescDB.hpp
- ProblemDescDB.cpp
13.158 ProcessApplicInterface Class Reference

Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.

Inheritance diagram for ProcessApplicInterface:

```
<table>
<thead>
<tr>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationInterface</td>
</tr>
<tr>
<td>ProcessApplicInterface</td>
</tr>
<tr>
<td>ProcessHandleApplicInterface</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
</tr>
<tr>
<td>SpawnApplicInterface</td>
</tr>
<tr>
<td>GridApplicInterface</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- `ProcessApplicInterface (const ProblemDescDB &problem_db)`
  *constructor*

- `~ProcessApplicInterface ()`
  *destructor*

Protected Member Functions

- `void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)`
  *Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.*

- `void derived_map_asynch (const ParamResponsePair &pair)`
  *Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.*

- `const StringArray & analysis_drivers () const`
  *retrieve the analysis drivers specification for application interfaces*

- `void file_cleanup () const`

- `void remove_params_results_files (const bfs::path &params_path, const bfs::path &results_path) const`
  *Remove (potentially autotagged for multiple programs) parameters and results files with passed root names.*

- `void autotag_files (const bfs::path &params_path, const bfs::path &results_path, const String &eval_id_tag) const`
  *Utility to automatically tag parameters and results files with passed root names (the files may already need per-program tagging)*

- `virtual void map_bookkeeping (pid_t pid, int fn_eval_id)=0`
  *bookkeeping of process and evaluation ids for asynchronous maps*

- `virtual pid_t create_evaluation_process (bool block_flag)=0`
Spawn the evaluation by managing the input filter, analysis drivers, and output filter. Called from `derived_map()` & `derived_map_asynch()`.

- **void synchronous_local_analyses** (int start, int end, int step)
  execute analyses synchronously on the local processor

- **void define_filenames** (const String &eval_id_tag)
  define modified filenames from user input by handling Unix temp file and optionally tagging with given eval_id_tag

- **void write_parameters_files** (const Variables &vars, const ActiveSet &set, const Response &response, const int id)
  write the parameters data and response request data to one or more parameters files (using one or more invocations of `write_parameters_file()`) in either standard or aprepro format

- **void read_results_files** (Response &response, const int id, const String &eval_id_tag)
  read the response object from one or more results files using full eval_id_tag passed

- **bfs::path get_workdir_name()**
  construct a work directory name (tmp or named), with optional tag

- **void prepare_process_environment()**
  set PATH, environment variables, and change directory prior to fork/system/spawn

- **void reset_process_environment()**
  reset PATH and current directory after system/spawn (workdir case)

### Protected Attributes

- **bool fileTagFlag**
  flags tagging of parameter/results files

- **bool fileSaveFlag**
  flags retention of parameter/results files

- **bool commandLineArgs**
  flag indicating use of passing of filenames as command line arguments to the analysis drivers and input/output filters

- **bool apreproFlag**
  flag indicating use of the APREPRO (the Sandia "A PRE PROcessor" utility) format for parameter files

- **unsigned short resultsFileFormat**
  results file format

- **bool multipleParamsFiles**
  flag indicating the need for separate parameters files for multiple analysis drivers

- **std::string iFilterName**
  the name of the input filter (input filter user specification)

- **std::string oFilterName**
  the name of the output filter (output filter user specification)

- **std::vector<String> programNames**
  the names of the analysis code programs (analysis drivers user specification)

- **std::string specifiedParamsFileName**
  the name of the parameters file from user specification

- **std::string paramsFileName**
  the parameters file name actually used (modified with tagging or temp files); only valid from define_filenames to write_parameters_files
- std::string paramsFileWritten
  actual, qualified name of the params file written, possibly with workdir
- std::string specifiedResultsFileName
  the name of the results file from user specification
- std::string resultsFileName
  the results file name actually used (modified with tagging or temp files); only valid from define_filenames to write_parameters_files
- std::string resultsFileWritten
  actual, qualified name of the results file written, possibly with workdir
- std::string fullEvalId
  complete evalIdTag, possibly including hierarchical tagging and final eval id, but not program numbers, for passing to write_parameters_files
- bool allowExistingResults
  by default analysis code interfaces delete results files if they exist; user may override with this flag and we’ll try to gather and only fork if needed
- std::map< int, PathTriple > fileNameMap
  Maps function evaluation ID to triples (parameters, results, and workdir) paths used in spawning function evaluations. Workdir will be empty if not created specifically for this eval.
- bool useWorkdir
  whether to use a work directory
- std::string workDirName
  work directory name, if specified...
- bool dirTag
  whether to tag the working directory
- bool dirSave
  whether dir_save was specified
- bfs::path curWorkdir
  active working directory for this evaluation; valid only from define_filenames to create_evaluation_process
- bfs::path createdDir
  non-empty if created for this eval; valid only from define_filenames to write_parameters_files
- StringArray linkFiles
  template directory (if specified)
- StringArray copyFiles
  template files (if specified)
- bool templateReplace
  whether to replace existing files

Private Member Functions

- void write_parameters_file (const Variables &vars, const ActiveSet &set, const Response &response, const std::string &prog, const std::vector< String > &an_comps, const std::string &params_fname)
  write the variables, active set vector, derivative variables vector, and analysis components to the specified parameters file in either standard or aprepro format
- void read_results_file (Response &response, const bfs::path &path, const int id)
  Open and read the results file at path, properly handling errors.
Private Attributes

- String2DArray analysisComponents
  
  the set of optional analysis components used by the analysis drivers (from the analysis_components interface specification)

13.158.1 Detailed Description

Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.

ProcessApplicInterface is subclassed for process handles or file completion testing.

13.158.2 Member Function Documentation

void file_cleanup ( ) const [protected], [virtual]

Remove any files and directories still referenced in the fileNameMap

  Reimplemented from Interface.

  References WorkdirHelper::concat_path(), ProcessApplicInterface::dirSave, ProcessApplicInterface::fileNameMap, ProcessApplicInterface::fileSaveFlag, ProcessApplicInterface::iFilterName, ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::programNames, and WorkdirHelper::recursive_remove().

void autotag_files ( const bfs::path & params_path, const bfs::path & results_path, const String & eval_id_tag ) const [protected]

Utility to automatically tag parameters and results files with passed root names (the files may already need per-program tagging)

  Move specified params and results files to unique tagged versions when needed

  References WorkdirHelper::concat_path(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::oFilterName, Interface::outputLevel, ProcessApplicInterface::programNames, WorkdirHelper::rename(), ProcessApplicInterface::specifiedParamsFileName, ProcessApplicInterface::specifiedResultsFileName, and ApplicationInterface::suppressOutput.

  Referenced by ProcessApplicInterface::read_results_files().

void synchronous_local_analyses ( int start, int end, int step ) [inline], [protected]

execute analyses synchronously on the local processor

  Execute analyses synchronously in succession on the local processor (start to end in step increments). Modeled after ApplicationInterface::synchronous_local_evaluations().

  References ApplicationInterface::synchronous_local_analysis().

  Referenced by ProcessHandleApplicInterface::create_evaluation_process().

void prepare_process_environment ( ) [protected]

set PATH, environment variables, and change directory prior to fork/system/spawn

  Guidance: environment (PATH, current directory) should be set immediately before Dakota spawns a process and reset immediately afterwards (except fork which never returns)

  References WorkdirHelper::change_directory(), ProcessApplicInterface::curWorkdir, Interface::outputLevel, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::resultsFileName, WorkdirHelper::set_environment(), WorkdirHelper::set_preferred_path(), and ProcessApplicInterface::useWorkdir.
CHAPTER 13. CLASS DOCUMENTATION

Referenced by SpawnApplicInterface::create_analysis_process(), ForkApplicInterface::create_analysis_process(), SysCallApplicInterface::spawn_analysis_to_shell(), SysCallApplicInterface::spawn_evaluation_to_shell(), SysCallApplicInterface::spawn_input_filter_to_shell(), and SysCallApplicInterface::spawn_output_filter_to_shell.

```cpp
void reset_process_environment() [protected]
```
reset PATH and current directory after system/spawn (workdir case)

Undo anything done prior to spawn

References Interface::outputLevel, WorkdirHelper::reset(), WorkdirHelper::startup_pwd(), and ProcessApplicInterface::useWorkdir.

Referenced by SpawnApplicInterface::create_analysis_process(), ForkApplicInterface::create_analysis_process(), SysCallApplicInterface::spawn_analysis_to_shell(), SysCallApplicInterface::spawn_evaluation_to_shell(), SysCallApplicInterface::spawn_input_filter_to_shell(), and SysCallApplicInterface::spawn_output_filter_to_shell.

```cpp
void read_results_file (Response & response, const bfs::path & path, const int id) [private]
```
Open and read the results file at path, properly handling errors.

Helper for read_results_files that opens the results file at results_path and reads it, handling various errors/exceptions.

References Dakota::abort_handler(), Response::read(), and ProcessApplicInterface::resultsFileFormat.

Referenced by ProcessApplicInterface::read_results_files().

The documentation for this class was generated from the following files:

- ProcessApplicInterface.hpp
- ProcessApplicInterface.cpp

### 13.159 ProcessHandleApplicInterface Class Reference

Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.

Inheritance diagram for ProcessHandleApplicInterface:

```
  Interface
    ApplicationInterface
      ProcessApplicInterface
        ProcessHandleApplicInterface
```

```
  ForkApplicInterface   SpawnApplicInterface
```

**Public Member Functions**

- **ProcessHandleApplicInterface (const ProblemDescDB &problem_db)**
  
  *constructor*
• ~ProcessHandleApplicInterface ()
  destructor

Protected Member Functions

• int synchronous_local_analysis (int analysis_id)
• void init_communicators_checks (int max_eval_concurrency)
• void set_communicators_checks (int max_eval_concurrency)
• void map_bookkeeping (pid_t pid, int fn_eval_id)
  bookkeeping of process and evaluation ids for asynchronous maps
• pid_t create_evaluation_process (bool block_flag)
• virtual pid_t create_analysis_process (bool block_flag, bool new_group)=0
  spawn a child process for an analysis component within an evaluation
• virtual size_t wait_local_analyses ()=0
  wait for asynchronous analyses on the local processor, completing at least one job
• virtual size_t test_local_analyses_send (int analysis_id)=0
  test for asynchronous analysis completions on the local processor and return results for any completions by sending messages
• virtual void join_evaluation_process_group (bool new_group)
  create (if new_group) and join the process group for async evaluations
• virtual void join_analysis_process_group (bool new_group)
  create (if new_group) and join the process group for async analyses
• virtual void evaluation_process_group_id (pid_t pgid)
  set evalProcGroupId
• virtual pid_t evaluation_process_group_id () const
  return evalProcGroupId
• virtual void analysis_process_group_id (pid_t pgid)
  set analysisProcGroupId
• virtual pid_t analysis_process_group_id () const
  return analysisProcGroupId
• void process_local_evaluation (PRPQueue &prp_queue, const pid_t pid)
  Common processing code used by {wait,test}_{local,evaluation}.
• void check_wait (pid_t pid, int status)
  check the exit status of a forked process and abort if an error code was returned
• void asynchronous_local_analyses (int start, int end, int step)
  execute analyses asynchronously on the local processor
• void serve_analyses_asynch ()
  serve the analysis scheduler and execute analysis jobs asynchronously
• void ifilter_argument_list ()
  set argList for execution of the input filter
• void ofilter_argument_list ()
  set argList for execution of the output filter
• void driver_argument_list (int analysis_id)
  set argList for execution of the specified analysis driver
• void create_command_arguments (boost::shared_array< const char * > &av, StringArray &driver_and_args)
  parse argList into argument array av suitable for passing to execvp, appending parameters and results filenames if
  requested by commandLineArgs

Protected Attributes
• std::map< pid_t, int > evalProcessIdMap
  map of fork process id’s to function evaluation id’s for asynchronous evaluations
• std::map< pid_t, int > analysisProcessIdMap
  map of fork process id’s to analysis job id’s for asynchronous analyses
• std::vector< std::string > argList
  an array of strings for use with execvp(const char *, char * const *). These are converted to an array of const
  char *’s in fork_program().

13.159.1 Detailed Description
Derived application interface class that spawns a simulation code using a separate process, receives a process
identifier, and communicates with the spawned process through files.
  ProcessHandleApplicInterface is subclassed for fork/execvp/waitpid (Unix) and spawnvp (Windows).

13.159.2 Constructor & Destructor Documentation
ProcessHandleApplicInterface ( const ProblemDescDB & problem_db ) [inline]
constructor
  argList sized 3 for [driver name, input file, output file]

13.159.3 Member Function Documentation
int synchronous_local_analysis ( int analysis_id ) [inline], [protected], [virtual]
This code provides the derived function used by ApplicationInterface:: serve_analyses_synch() as well as a con-
venience function for ProcessHandleApplicInterface::synchronous_local_analyses() below.
  Reimplemented from ApplicationInterface.
  References ProcessHandleApplicInterface::create_analysis_process(), and ProcessHandleApplicInterface::driver-
 _argument_list().

void init_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
No derived interface plug-ins, so perform construct-time checks. However, process init issues as warnings since
some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used.
  Reimplemented from ApplicationInterface.
  References ApplicationInterface::check_multiprocessor_analysis(), and ApplicationInterface::check_multiprocessor-
  _asynchronous().

void set_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
Process run-time issues as hard errors.
  Reimplemented from ApplicationInterface.
  References Dakota::abort_handler(), ApplicationInterface::check_multiprocessor_analysis(), and Application-
  Interface::check_multiprocessor_asynchronous().
**pid_t create_evaluation_process ( bool block_flag ) [protected], [virtual]**

Manage the input filter, 1 or more analysis programs, and the output filter in blocking or nonblocking mode as governed by block_flag. In the case of a single analysis and no filters, a single fork is performed, while in other cases, an initial fork is reforked multiple times. Called from derived_map() with block_flag == BLOCK and from derived_map_asynch() with block_flag == FALL_THROUGH. Uses create_analysis_process() to spawn individual program components within the function evaluation.

Implements ProcessApplicInterface.

References Dakota::abort_handler(), ProcessHandleApplicInterface::analysis_process_group_id(), ApplicationInterface::analysisServerId, ApplicationInterface::asynchLocalAnalysisConcurrency, ApplicationInterface::asynchLocalAnalysisFlag, ProcessHandleApplicInterface::asynchronous_local_analyses(), ParallelLibrary::barrier_e(), ProcessApplicInterface::commandLineArgs, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ApplicationInterface::eaDedMasterFlag, ApplicationInterface::evalCommRank, ApplicationInterface::evalCommSize, ProcessHandleApplicInterface::evalProcessIdMap, ProcessHandleApplicInterface::evaluation_process_group_id(), ProcessHandleApplicInterface::ifilter_argument_list(), ProcessApplicInterface::iFilterName, ProcessHandleApplicInterface::join_evaluation_process_group(), ApplicationInterface::master_dynamic_schedule_analyses(), ProcessApplicInterface::multipleParamsFiles, ApplicationInterface::numAnalysisDrivers, ApplicationInterface::numAnalysisServers, ProcessHandleApplicInterface::ofilter_argument_list(), ProcessApplicInterface::oFilterName, ApplicationInterface::parallelLib, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, ProcessHandleApplicInterface::serve_analyses_asynch(), ApplicationInterface::serve_analyses_synch(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::synchronous_local_analyses().

**void check_wait ( pid_t pid, int status ) [protected]**

check the exit status of a forked process and abort if an error code was returned.

Check to see if the process terminated abnormally (WEXITED(status)==0) or if either execvp or the application returned a status code of -1 (WEXITED(status)!=0 && (signed char)WEXITSTATUS(status)==-1). If one of these conditions is detected, output a failure message and abort. Note: the application code should not return a status code of -1 unless an immediate abort of dakota is wanted. If for instance, failure capturing is to be used, the application code should write the word "FAIL" to the appropriate results file and return a status code of 0 through exit().

Referenced by ForkApplicInterface::create_analysis_process(), SpawnApplicInterface::test_local_analyses_send(), SpawnApplicInterface::test_local_evaluations(), ForkApplicInterface::wait(), SpawnApplicInterface::wait_local_analyses(), and SpawnApplicInterface::wait_local_evaluations().

**void asynchronous_local_analyses ( int start, int end, int step ) [protected]**

execute analyses asynchronously on the local processor.

Schedule analyses asynchronously on the local processor using a dynamic scheduling approach (start to end in step increments). Concurrency is limited by asynchLocalAnalysisConcurrency. Modeled after ApplicationInterface::asynchronous_local_evaluations(). NOTE: This function should be elevated to ApplicationInterface if and when another derived interface class supports asynchronous local analyses.

Referenced by ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ApplicationInterface::numAnalysisDrivers, and ProcessHandleApplicInterface::wait_local_analyses().

Referenced by ProcessHandleApplicInterface::create_evaluation_process().
CHAPTER 13. CLASS DOCUMENTATION

```cpp
void serve_analyses_asynch() [protected]
```

serve the analysis scheduler and execute analysis jobs asynchronously

This code runs multiple asynch analyses on each server. It is modeled after ApplicationInterface::serve_evaluations_asynch. NOTE: This fn should be elevated to ApplicationInterface if and when another derived interface class supports hybrid analysis parallelism.

References Dakota::abort_handler(), ProcessHandleApplicInterface::analysisProcessIdMap, ApplicationInterface::asynchLocalAnalysisConcurrency, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ParallelLibrary::irecv_ea(), ApplicationInterface::numAnalysisDrivers, ApplicationInterface::parallelLib, ParallelLibrary::recv_ea(), ParallelLibrary::test(), and ProcessHandleApplicInterface::test_local_analyses_send().

Referenced by ProcessHandleApplicInterface::create_evaluation_process().

```cpp
void create_command_arguments( boost::shared_array< const char∗ > & av, StringArray & driver_and_args ) [protected]
```

parse argList into argument array av suitable for passing to execvp, appending parameters and results filenames if requested by commandLineArgs

This function will split the analysis command in argList[0] based on whitespace, but preserve spaces within quoted strings, such that quoted strings can be passed as single command arguments. NOTE: This function allocates memory in av that might be implicitly freed when the child exits (control never returns to caller). driver_and_args needs to be a return argument because av will contain pointers into its c_str()’s when done.

References ProcessHandleApplicInterface::argList, ProcessApplicInterface::commandLineArgs, and WorkdirHelper::tokenize_driver().

Referenced by SpawnApplicInterface::create_analysis_process(), and ForkApplicInterface::create_analysis_process().

The documentation for this class was generated from the following files:

- ProcessHandleApplicInterface.hpp
- ProcessHandleApplicInterface.cpp

13.160 ProgramOptions Class Reference

ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag()

Public Member Functions

- ProgramOptions ()
  
  default constructor (needed for default environment ctors and could be used by library clients to late update data)

- ProgramOptions (int world_rank)
  
  constructor that accepts world rank to help with I/O control; allows default constructed ProgramOptions to get rank in library mode

- ProgramOptions (int argc, char *argv[], int world_rank)
  
  standard constructor that uses a CommandLineHandler to parse user options

- const String & input_file () const
  
  Dakota input file base name (no tag)

- const String & input_string () const
alternate Dakota input string literal

- bool echo_input () const
  is input echo specified?
- const String & parser_options () const
  (deprecated) NIDR parser options
- String output_file () const
  output (user-provided or default) file base name (no tag)
- const String & error_file () const
  error file base name (no tag)
- const String & exit_mode () const
  behavior of abort_handler (throw or exit)
- const String & read_restart_file () const
  restart file base name (no tag)
- size_t stop_restart_evals () const
  eval ID at which to stop reading restart
- String write_restart_file () const
  write restart (user-provided or default) file base name (no tag)
- bool help () const
  is help mode active?
- bool version () const
  is version mode active?
- bool check () const
  is check mode active?
- bool pre_run () const
  is pre-run mode active?
- bool run () const
  is run mode active?
- bool post_run () const
  is post-run mode active?
- bool user_modes () const
  are any non-default, user-specified run modes active?
- const String & pre_run_input () const
  filename for pre-run input
- const String & pre_run_output () const
  filename for pre-run output
- const String & run_input () const
  filename for run input
- const String & run_output () const
  filename for run output
- const String & post_run_input () const
  filename for post-run input
- const String & post_run_output () const
  filename for post-run output
• unsigned int **pre_run_output_format** () const
  tabular format for pre-run output
• unsigned int **post_run_input_format** () const
  tabular format for post-run input
• bool **proceed_to_instantiate** () const
  whether steps beyond help/version are requested (instantiation required)
• bool **proceed_to_run** () const
  Whether steps beyond check are requested.
• bool **user_stdout_redirect** () const
  whether the user/client code requested a redirect of stdout
• bool **user_stderr_redirect** () const
  whether the user/client code requested a redirect of stderr
• void **world_rank** (int world_rank)
  set the world rank to govern early conditional output
• void **input_file** (const String &in_file)
  set Dakota input file base name (no tag)
• void **input_string** (const String &in_string)
  set alternate Dakota input string literal
• void **echo_input** (bool echo_flag)
  set whether to echo input to output
• void **exit_mode** (const String &mode)
  set behavior for abort_handler
• void **output_file** (const String &out_file)
  set base file name for Dakota output
• void **error_file** (const String &err_file)
  set base file name for Dakota errors
• void **read_restart_file** (const String &read_rst)
  set base file name for restart file from which to read
• void **stop_restart_evals** (size_t stop_rst)
  set eval ID at which to stop reading restart
• void **write_restart_file** (const String &write_rst)
  set base file name for restart file to write
• void **help** (bool help_flag)
  set true to print help information and exit
• void **version** (bool version_flag)
  set true to print version information and exit
• void **check** (bool check_flag)
  set true to check input and instantiate objects, then exit
• void **pre_run** (bool pre_run_flag)
  set to enable/disable pre-run phase
• void **run** (bool run_flag)
  set to enable/disable run phase
• void **post_run** (bool post_run_flag)
set to enable/disable post-run phase

- void pre_run_input (const String &pre_run_in)
  Specify the pre-run phase input filename.
- void pre_run_output (const String &pre_run_out)
  Specify the pre-run phase output filename.
- void run_input (const String &run_in)
  Specify the run phase input filename.
- void run_output (const String &run_out)
  Specify the run phase output filename.
- void post_run_input (const String &post_run_in)
  Specify the post-run phase input filename.
- void post_run_output (const String &post_run_out)
  Specify the post-run phase output filename.
- void parse (const ProblemDescDB &problem_db)
  Extract environment options from ProblemDescDB.
- void read (MPIUnpackBuffer &s)
  helper function for reading some class data from MPI buffer
- void write (MPIPackBuffer &s) const
  helper function for writing some class data to MPI buffer

Private Member Functions

- void parse_environment_options ()
  any environment variables affecting global behavior get read here
- void manage_run_modes (const CommandLineHandler &clh)
  retrieve run mode options from the CLH
- void split_filenames (const char *filenames, std::string &input_filename, std::string &output_filename)
  manage pre/run/post filenames
- void validate ()
  verify consistency of user settings (helpful for library mode especially)
- void validate_run_modes ()
  validate user run modes and set userModesFlag
- void set_option (const ProblemDescDB &problem_db, const String &db_name, String &data_member)
  retrieve environment.<db_name> from the problem db and update data_member, warning if needed

Private Attributes

- int worldRank
  cache the world rank to help with conditional output
- String inputFile
  Dakota input file name, e.g., "dakota.in".
- String inputString
  alternate input means for library clients: input string (mutually exclusive with input file)
- bool echoInput
whether to echo client’s input file at parse

- **String parserOptions**
  
  Deprecated option for NIDR parser options.

- **String exitMode**
  
  Abort or throw on error.

- **String outputFile**
  
  Dakota output base file name, e.g., "dakota.out".

- **String errorFile**
  
  Dakota error base file name, e.g., "dakota.err".

- **String readRestartFile**
  
  e.g., "dakota.old.rst"

- **size_t stopRestartEvals**
  
  eval number at which to stop restart read

- **String writeRestartFile**
  
  e.g., "dakota.new.rst"

- **bool helpFlag**
  
  whether to print help message and exit

- **bool versionFlag**
  
  whether to print version message and exit

- **bool checkFlag**
  
  flags invocation with command line option -check

- **bool preRunFlag**
  
  flags invocation with command line option -pre_run

- **bool runFlag**
  
  flags invocation with command line option -run

- **bool postRunFlag**
  
  flags invocation with command line option -post_run

- **bool userModesFlag**
  
  whether any user run modes are active

- **String preRunInput**
  
  filename for pre_run input

- **String preRunOutput**
  
  filename for pre_run output

- **String runInput**
  
  filename for run input

- **String runOutput**
  
  filename for run output

- **String postRunInput**
  
  filename for post_run input

- **String postRunOutput**
  
  filename for post_run output

- **unsigned short preRunOutputFormat**
  
  tabular format for pre_run output

- **unsigned short postRunInputFormat**
  
  tabular format for post_run input
13.160.1 Detailed Description

ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag()

13.160.2 Member Function Documentation

void split_filenames ( const char ∗ filenames, std::string & input_filename, std::string & output_filename )
[private]
manage pre/run/post filenames
Tokenize colon-delimited input and output filenames, returns unchanged strings if tokens not found.
Referenced by ProgramOptions::manage_run_modes().
The documentation for this class was generated from the following files:

- ProgramOptions.hpp
- ProgramOptions.cpp

13.161 PStudyDACE Class Reference

Base class for managing common aspects of parameter studies and design of experiments methods.
Inheritance diagram for PStudyDACE:

![Inheritance Diagram](image)

Public Member Functions

- bool resize ()
  
  reinitalizes iterator based on new variable size

Protected Member Functions

- PStudyDACE (ProblemDescDB &problem_db, Model &model)
  
  constructor
- PStudyDACE (unsigned short method_name, Model &model)
  
  alternate constructor for instantiations “on the fly”
- ~PStudyDACE ()
  
  destructor
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  
  print the final iterator results
• void volumetric_quality (int ndim, int num_samples, double *sample_points)

  Calculation of volumetric quality measures.

Protected Attributes

• SensAnalysisGlobal pStudyDACE::SensGlobal
  initialize statistical post processing
• bool volQualityFlag
  flag which specifies evaluation of volumetric quality measures
• bool varBasedDecompFlag
  flag which specifies calculating variance based decomposition sensitivity analysis metrics

Private Attributes

• double chiMeas
  quality measure
• double dMeas
  quality measure
• double hMeas
  quality measure
• double tauMeas
  quality measure

Additional Inherited Members

13.16.1.1 Detailed Description

Base class for managing common aspects of parameter studies and design of experiments methods.

The PStudyDACE base class manages common data and functions, such as those involving the best solutions located during the parameter set evaluations or the printing of final results.

13.16.1.2 Member Function Documentation

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Analyzer.

References PStudyDACE::chiMeas, Analyzer::compactMode, Model::continuous_variable_labels(), SensAnalysisGlobal::correlations_computed(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), Model::discrete_string_variable_labels(), PStudyDACE::dMeas, PStudyDACE::hMeas, Iterator::iteratedModel, Analyzer::numLSqTerms, Analyzer::numObjFns, SensAnalysisGlobal::print_correlations(), Analyzer::print_results(), Analyzer::print_sobol_indices(), PStudyDACE::pStudyDACE::SensGlobal, Model::response_labels(), PStudyDACE::tauMeas, PStudyDACE::varBasedDecompFlag, and PStudyDACE::volQualityFlag.
void volumetric_quality ( int ndim, int num_samples, double *sample_points ) [protected]

Calculation of volumetric quality measures.
Calculation of volumetric quality measures developed by FSU.
References PStudyDACE::chiMeas, PStudyDACE::dMeas, PStudyDACE::hMeas, and PStudyDACE::tauMeas.
Referenced by FSUDesignCompExp::get_parameter_sets(), and DDACEDesignCompExp::get_parameter_sets().
The documentation for this class was generated from the following files:

- DakotaPStudyDACE.hpp
- DakotaPStudyDACE.cpp

13.162 PSUADEDesignCompExp Class Reference

Wrapper class for the PSUADE library.

Inheritance diagram for PSUADEDesignCompExp:

```
PSUADEDesignCompExp
  |                  ...
  |                  |
  |                  |
  |                  | Iterator
  |                  |
  | Analyzer         |
  | PStudyDACE       |
  | PSUADEDesignCompExp
```

**Public Member Functions**

- **PSUADEDesignCompExp** (ProblemDescDB &problem_db, Model &model)
  
  *primary constructor for building a standard DACE iterator*

- **~PSUADEDesignCompExp** ()
  
  *destructor*

- bool resize ()
  
  *reinitializes iterator based on new variable size*

**Protected Member Functions**

- void pre_run ()
  
  *pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- void post_input ()
  
  *read tabular data for post-run mode*

- void core_run ()
  
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

- void post_run (std::ostream &s)
post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

- int num_samples () const
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator to use at least min_samples
- unsigned short sampling_scheme () const
  return sampling name
- void vary_pattern (bool pattern_flag)
  sets varyPattern in derived classes that support it
- void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim + num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.

Private Member Functions

- void enforce_input_rules ()
  enforce sanity checks/modifications for the user input specification

Private Attributes

- int samplesSpec
  initial specification of number of samples
- int numSamples
  current number of samples to be evaluated
- const UShortArray & varPartitionsSpec
  number of partitions in each variable direction
- int numPartitions
  number of partitions to pass to PSUADE (levels = partitions + 1)
- bool allDataFlag
  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()
- size_t numDACERuns
  counter for number of executions for this object
- bool varyPattern
  flag for generating a sequence of seed values within multiple get_parameter_sets() calls so that the sample sets are not repeated, but are still repeatable
- const int seedSpec
  the user seed specification for the random number generator (allows repeatable results)
- int randomSeed
  current seed for the random number generator
Additional Inherited Members

13.162.1 Detailed Description

Wrapper class for the PSUADE library.

The **PSUADEDesignCompExp** class provides a wrapper for PSUADE, a C++ design of experiments library from Lawrence Livermore National Laboratory. Currently this class only includes the PSUADE Morris One-at-a-time (MOAT) method to uniformly sample the parameter space spanned by the active bounds of the current Model. It returns all generated samples and their corresponding responses as well as the best sample found.

13.162.2 Constructor & Destructor Documentation

**PSUADEDesignCompExp ( ProblemDescDB & problem_db, Model & model )**

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Iterator::method_string(), Iterator::method-Name, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, and PSUADEDesignCompExp::numSamples.

13.162.3 Member Function Documentation

**void pre_run ( ) [protected], [virtual]**

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent's pre_run(), if implemented, typically **before** performing its own implementation steps.

Reimplemented from Analyzer.

References PSUADEDesignCompExp::enforce_input_rules(), PSUADEDesignCompExp::get_parameter_sets(), Iterator::iteratedModel, and Analyzer::pre_run().

**void core_run ( ) [protected], [virtual]**

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, Analyzer::numLSqTerms, and Analyzer::numObjFns.

**void post_run ( std::ostream & s ) [protected], [virtual]**

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent's post_run(), typically **after** performing its own implementation steps.

Reimplemented from Analyzer.
References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), PSUADEDesignCompExp::enforce_input_rules(), Iterator::iterated_Model, Analyzer::numContinuousVars, Analyzer::numFunctions, PSUADEDesignCompExp::numSamples, and Analyzer::post_run().

```cpp
int num_samples( ) const [inline], [protected], [virtual]
```

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References PSUADEDesignCompExp::numSamples.

```cpp
void enforce_input_rules( ) [private]
```

enforce sanity checks/modifications for the user input specification

Users may input a variety of quantities, but this function must enforce any restrictions imposed by the sampling algorithms.

References Analyzer::numContinuousVars, PSUADEDesignCompExp::numPartitions, PSUADEDesignCompExp::numSamples, and PSUADEDesignCompExp::varPartitionsSpec.

Referenced by PSUADEDesignCompExp::post_input(), PSUADEDesignCompExp::post_run(), and PSUADEDesignCompExp::pre_run().

The documentation for this class was generated from the following files:

- PSUADEDesignCompExp.hpp
- PSUADEDesignCompExp.cpp

### 13.163 PythonInterface Class Reference

Inheritance diagram for PythonInterface:

```
  Interface
    ApplicationInterface
      DirectApplicInterface
        PythonInterface
```

**Public Member Functions**

- **PythonInterface (const ProblemDescDB &problem_db)**
  - *constructor*
- **~PythonInterface ()**
  - *destructor*
Protected Member Functions

- virtual int derived_map_ac (const String &ac_name)
  
  *execute an analysis code portion of a direct evaluation invocation*

- int python_run (const String &ac_name)
  
  *direct interface to Python via API. BMA 07/02/07*

- template<class ArrayT, class Size>
  bool python_convert_int (const ArrayT &src, Size size, PyObject **dst)
  
  *convert arrays of integer types to Python list or numpy array*

- bool python_convert (const RealVector &src, PyObject **dst)
  
  *convert RealVector to Python list or numpy array*

- bool python_convert (const RealVector &c_src, const IntVector &di_src, const RealVector &dr_src, PyObject **dst)
  
  *convert RealVector + IntVector + RealVector to Python mixed list or numpy double array*

- template<class StringArrayT>
  bool python_convert_strlist (const StringArrayT &src, PyObject **dst)
  
  *convert labels*

- bool python_convert (const StringMultiArray &c_src, const StringMultiArray &di_src, const StringMultiArray &dr_src, PyObject **dst)
  
  *convert all labels to single list*

- bool python_convert (PyObject *pyv, RealVector &rv, const int &dim)
  
  *convert python [list of int or float] or [numpy array of double] to RealVector (for fns)*

- bool python_convert (PyObject *pyv, double *rv, const int &dim)
  
  *convert python [list of int or float] or [numpy array of double] to double[], for use as helper in converting gradients*

- bool python_convert (PyObject *pym, RealMatrix &rm)
  
  *convert python [list of lists of int or float] or [numpy array of dbl] to RealMatrix (for gradients)*

- bool python_convert (PyObject *pym, RealSymMatrix &rm)
  
  *convert python [list of lists of int or float] or [numpy array of dbl] to RealMatrix (used as helper in Hessian conversion)*

- bool python_convert (PyObject *pyma, RealSymMatrixArray &rma)
  
  *convert python [list of lists of lists of int or float] or [numpy array of double] to RealSymMatrixArray (for Hessians)*

Protected Attributes

- bool userNumpyFlag
  
  *whether the user requested numpy data structures in the input file*

- bool ownPython
  
  *true if this class created the interpreter instance*

13.163.1 Detailed Description

Specialization of DirectApplicInterface to link to Python analysis drivers. Includes convenience functions to map data to/from Python.
13.163.2 Member Function Documentation

```c
int derived_map_ac ( const String & ac_name ) [protected], [virtual]
```
execute an analysis code portion of a direct evaluation invocation
Python specialization of derived analysis components.
Reimplemented from DirectApplicInterface.
References ApplicationInterface::analysisServerId, and PythonInterface::python_run().

```c
bool python_convert_int ( const ArrayT & src, Size sz, PyObject ** dst ) [protected]
```
convert arrays of integer types to Python list or numpy array
convert all integer array types including IntVector, ShortArray, and SizetArray to Python list of ints or numpy
array of ints
References PythonInterface::userNumpyFlag.
Referenced by PythonInterface::python_run().

The documentation for this class was generated from the following files:
- PythonInterface.hpp
- PythonInterface.cpp

13.164 QuesoJointPdf\(< V, M >\) Class Template Reference

**Dakota** specialization of QUESO generic joint PDF.
Inherits BaseJointPdf\(< V, M >\).

**Public Member Functions**
- `QuesoJointPdf (const char *prefix, const QUESO::VectorSet< V, M > &domainSet, NonDQUESOBayesCalibration *nond_queso_ptr)`
  Default constructor.
- `virtual ~QuesoJointPdf ()`
  Destructor.
- `double actualValue (const V &domainVector, const V *domainDirection, V *gradVector, M *hessianMatrix, V *hessianEffect) const`
  Actual value of the PDF (scalar function).
- `double lnValue (const V &domainVector, const V *domainDirection, V *gradVector, M *hessianMatrix, V *hessianEffect) const`
  Logarithm of the value of the function.
- `double computeLogOfNormalizationFactor (unsigned int numSamples, bool m_logOfNormalizationFactor) const`
  Computes the logarithm of the normalization factor.
- `void distributionMean (V &meanVector) const`
  Mean value of underlying random variable.
- `void distributionVariance (M &covMatrix) const`

**Private Attributes**
- `NonDQUESOBayesCalibration * nondQUESOInstance`
13.164.1 Detailed Description

template<class V, class M> class Dakota::QuesoJointPdf< V, M >

Dakota specialization of QUESO generic joint PDF.

13.164.2 Constructor & Destructor Documentation

QuesoJointPdf ( const char * prefix, const QUESO::VectorSet< V, M > & domainSet,
NonDQUESOBayesCalibration * nond_queso_ptr )

Default constructor.

Instantiates an object of the class, i.e. a scalar function, given a prefix and its domain.
The documentation for this class was generated from the following file:

- NonDQUESOBayesCalibration.cpp

13.165 QuesoVectorRV< V, M > Class Template Reference

Dakota specialization of QUESO vector-valued random variable.

Inherits BaseVectorRV< V, M >.

Public Member Functions

- QuesoVectorRV ( const char *prefix, const QUESO::VectorSet< V, M > &imageSet, NonDQUESOBayesCalibration *nond_queso_ptr )

  Default constructor.

- virtual ~QuesoVectorRV ()

  Virtual destructor.

- void print ( std::ostream &os ) const

  TODO: Prints the vector RV (required pure virtual).

13.165.1 Detailed Description

template<class V, class M> class Dakota::QuesoVectorRV< V, M >

Dakota specialization of QUESO vector-valued random variable.

13.165.2 Constructor & Destructor Documentation

QuesoVectorRV ( const char * prefix, const QUESO::VectorSet< V, M > & imageSet,
NonDQUESOBayesCalibration * nond_queso_ptr )

Default constructor.

Constructs a generic queso vector RV, given a prefix and the image set of the vector RV.

The documentation for this class was generated from the following file:

- NonDQUESOBayesCalibration.cpp
13.166 RandomFieldModel Class Reference

Random field model, capable of generating and then forward propagating.

Inheritance diagram for RandomFieldModel:

```
Model
|
RecastModel
|
RandomFieldModel
```

Public Member Functions

- **RandomFieldModel (ProblemDescDB &problem_db)**
  
  Problem database constructor.

- **~RandomFieldModel ()**
  
  Destructor

- **bool initialize_mapping (ParLevLIter pl_iter)**
  
  For KL models, the model is augmented with the random coeffs of the KL.

- **bool finalize_mapping ()**
  
  Finalize model mapping, returns true if the variables size has changed.

- **bool mapping_initialized () const**
  
  Return true if mapping has been fully initialized, false otherwise.

- **bool resize_pending () const**
  
  Return true if a potential resize is still pending, such that sizing-based initialization should be deferred.

Protected Member Functions

- **Model get_sub_model (ProblemDescDB &problem_db)**
  
  Retrieve the sub-Model from the DB to pass up the constructor chain.

- **void init_dace_iterator (ProblemDescDB &problem_db)**
  
  Initialize the RF-generating sampler.

- **void validate_inputs ()**
  
  Validate the build controls and set defaults.

- **void get_field_data ()**
  
  Source data generation: get the field data either from file or simulation by running the DACE Iterator. Populates rfBuildData.

- **void identify_field_model ()**
  
  Generate field representation: generate a KL or PCA/GP.

- **void rf_suite_identify_field_model ()**
  
  Generate field representation: utilize RF Suite.

- **void initialize_recast ()**
  
  Initialize the base class RecastModel with reduced space variable sizes.
13.166. RANDOMFIELDMODEL CLASS REFERENCE

- **SizetArray variables.resize ();**
  
  Create a variables components totals array with the reduced space size for continuous variables.
- **void initialize_rf_coeffs ();**
  
  For KL models, augment the subModel’s uncertain variables with additional N(0,1) variables; set up AleatoryDist-Params for the N(0,1)’s.
- **void derived_evaluate (const ActiveSet &set)**
  
  generate a random field realization, then evaluate the submodel
- **void derived_evaluate_nowait (const ActiveSet &set)**
  
  generate a random field realization, then evaluate the submodel (asynch)
- **void generate_kl_realization ();**
  
  generate a KL realization and write to file
- **void generate_pca_gp_realization ();**
  
  generate a PCA/GP realization and write to file
- **void write_field (const RealVector &field_prediction);**
  
  write a field realization to console and file

### Static Protected Member Functions

- **static void vars_mapping (const Variables &recast Xi_vars, Variables &sub_model_X_vars);**
  
  map the active continuous recast variables to the active submodel variables (linear transformation)
- **static void set_mapping (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set);**
  
  map the inbound ActiveSet to the sub-model (map derivative variables)

### Protected Attributes

- **size_t numFunctions**
- **String rfDataFilename**
  
  name of the data file with RF build data
- **size_t numObservations**
  
  rows of data matrix
- **IntVector fieldLengths**
  
  column partitions of data matrix
- **RealMatrix rfBuildData**
  
  data matrix with realizations of the random field to approximate
- **RealMatrix rfBuildVars**
  
  matrix of samples used to build the RF data
- **Iterator daceIterator**
  
  String dataDirectoryBasename;`
- **unsigned short expansionForm**
  
  unsigned short analyticCovForm;`
- **unsigned short covarianceForm**
  
  form of the analytic covariance function
- **int requestedReducedRank**
current approximation of system rank

- Real percentVariance
  fraction of energy to capture
- int actualReducedRank
  command to run RF Suite
- ReducedBasis rfBasis
  reduced basis representation (for KL or PCA case)
- int fieldRealizationId
  counter for RF Suite
- std::vector< Approximation > gpApproximations
  approximate models used to map the uncertain vars through the PCA approx
- bool mappingInitialized
  the index of the active metaiterator-iterator parallelism level (corresponding to ParallelConfiguration::miPLIers) used at runtime

Static Protected Attributes

- static RandomFieldModel * rfInstance
  static pointer to this class for use in static callbacks

Additional Inherited Members

13.166.1 Detailed Description

Random field model, capable of generating and then forward propagating.

Specialization of a RecastModel that optionally identifies an approximate random field model during build phase and creates a RecastModel capable of performing forward UQ including the field and auxiliary uncertain variables reduced space. This RandomFieldModel wraps the random field propagation model (not the RF-generating model)

13.166.2 Member Function Documentation

bool initialize_mapping ( ParLevLIter pl_iter ) [virtual]

for KL models, the model is augmented with the random coeffs of the KL

May eventually take on init_comms and related operations. Also may want ide of build/update like DataFitSurrModel, eventually.

Reimplemented from Model.

References RandomFieldModel::covarianceForm, Model::estimate_message_lengths(), RandomFieldModel::expansionForm, RandomFieldModel::fieldRealizationId, RandomFieldModel::get_field_data(), RandomFieldModel::identify_field_model(), RecastModel::initialize_mapping(), RandomFieldModel::initialize_recast(), RandomFieldModel::initialize_rf_coeffs(), RandomFieldModel::mappingInitialized, and RandomFieldModel::rf_suite_identify_field_model().
void get_field_data ( ) [protected]

Source data generation: get the field data either from file or simulation by running the DACE Iterator. Populates rfBuildData.

   References Iterator::all_responses(), Iterator::all_samples(), Dakota::copy_data(), Model::cv(), RandomFieldModel::daceIterator, RandomFieldModel::expansionForm, Iterator::is_null(), Iterator::num_samples(), RandomFieldModel::rfBuildData, RandomFieldModel::rfBuildVars, Iterator::run(), and RecastModel::subModel.

   Referenced by RandomFieldModel::initialize_mapping().

void rf_suite_identify_field_model ( ) [protected]

Generate field representation: utilize RF Suite.

   Alternative to below function when using RFSuite.

   References RandomFieldModel::actualReducedRank, and RandomFieldModel::requestedReducedRank.

   Referenced by RandomFieldModel::initialize_mapping().

void initialize_recast ( ) [protected]

Initialize the base class RecastModel with reduced space variable sizes.

   Initialize the recast model to augment the uncertain variables with actualReducedRank additional N(0,1) variables, with no response function mapping (for now).

   References RandomFieldModel::actualReducedRank, Model::aleatory_distribution_parameters(), Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Response::function_gradients(), Response::function_hessians(), RecastModel::init_maps(), RecastModel::init_sizes(), Model::num_functions(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), RandomFieldModel::rfmInstance, RandomFieldModel::set_mapping(), RecastModel::subModel, RandomFieldModel::variables_resize(), and RandomFieldModel::vars_mapping().

   Referenced by RandomFieldModel::initialize_mapping().

SizetArray variables_resize ( ) [protected]

Create a variables components totals array with the reduced space size for continuous variables.

   Create a variables components totals array with the reduced space size for continuous variables TODO: augment normal uncVars for KL case.

   References RandomFieldModel::actualReducedRank, SharedVariablesData::components_totals(), Model::current_variables(), RandomFieldModel::expansionForm, Variables::shared_data(), RecastModel::subModel, and Dakota::svd().

   Referenced by RandomFieldModel::initialize_recast().

void initialize_rf_coeffs ( ) [protected]

For KL models, augment the subModel’s uncertain variables with additional N(0,1) variables; set up AleatoryDistParams for the N(0,1)’s.

   Initialize the aleatory dist params for the KL coeffs

   References RandomFieldModel::actualReducedRank, Model::aleatory_distribution_parameters(), Variables::continuous_variable_label(), Model::continuous_variable_labels(), Model::currentVariables, RandomFieldModel::expansionForm, and RecastModel::subModel.

   Referenced by RandomFieldModel::initialize_mapping().
void vars_mapping ( const Variables & recast_xi_vars, Variables & sub_model_x_vars ) [static], [protected]

map the active continuous recast variables to the active submodel variables (linear transformation)
map the active continuous recast variables to the active submodel variables

References Variables::active_variables(), RandomFieldModel::actualReducedRank, Model::aleatory_distribution_parameters(), Variables::continuous_variables(), Model::cv(), Variables::discrete_int_variables(), Model::discrete_int_variables(), Variables::discrete_real_variables(), Model::discrete_real_variables(), Variables::discrete_string_variables(), Model::discrete_string_variables(), RandomFieldModel::expansionForm, RandomFieldModel::rfm_Instance, and RecastModel::subModel.

Referenced by RandomFieldModel::initialize_recast().

13.166.3 Member Data Documentation

**Iterator daceIterator** [protected]

String dataDirectoryBasename;

DACE Iterator to evaluate the RF generating model

Referenced by RandomFieldModel::get_field_data(), RandomFieldModel::init_dace_iterator(), and RandomFieldModel::validate_inputs().

**unsigned short expansionForm** [protected]

unsigned short analyticCovForm;

form of the RF representation (KL, PCA, ICA)

Referenced by RandomFieldModel::derived_evaluate(), RandomFieldModel::derived_evaluate_nowait(), RandomFieldModel::get_field_data(), RandomFieldModel::identify_field_model(), RandomFieldModel::initialize_mapping(), RandomFieldModel::initialize_rf_coeffs(), RandomFieldModel::resize_pending(), RandomFieldModel::variables_resize(), and RandomFieldModel::vars_mapping().

**int actualReducedRank** [protected]

command to run RF Suite

number of bases retained in decomposition

Referenced by RandomFieldModel::generate_kl_realization(), RandomFieldModel::generate_pca_gp_realization(), RandomFieldModel::identify_field_model(), RandomFieldModel::initialize_recast(), RandomFieldModel::initialize_rf_coeffs(), RandomFieldModel::rf_suite_identify_field_model(), RandomFieldModel::variables_resize(), and RandomFieldModel::vars_mapping().

**RandomFieldModel * rfmInstance** [static], [protected]

static pointer to this class for use in static callbacks

initialization of static needed by RecastModel

Referenced by RandomFieldModel::initialize_recast(), RandomFieldModel::RandomFieldModel(), and RandomFieldModel::vars_mapping().

**bool mappingInitialized** [protected]

the index of the active metaiterator-iterator parallelism level (corresponding to ParallelConfiguration::miPLIters)
used at runtime

track use of initialize_mapping() and finalize_mapping()
Referenced by RandomFieldModel::finalize_mapping(), RandomFieldModel::initialize_mapping(), RandomFieldModel::mapping_initialized(), and RandomFieldModel::resize_pending().

The documentation for this class was generated from the following files:

- RandomFieldModel.hpp
- RandomFieldModel.cpp

### 13.167 RecastModel Class Reference

Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

Inheritance diagram for RecastModel:

![Inheritance Diagram](image)

**Public Member Functions**


  * standard (full) constructor; assumes provided sizes and map functions are final and constructs all member data

- **RecastModel** (const Model &sub_model, const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset, short recast_resp_order)

  * alternate constructor; uses provided sizes to construct Variables, Response and Constraints so Model can be passed to an Iterator; requires subsequent init_map() call.

- **RecastModel (ProblemDescDB &problem_db, const Model &sub_model)**

  * Problem DB-based ctor, e.g., for use in subspace model; assumes mappings to be initialized later; only initializes based on sub-model.

- **RecastModel** (const Model &sub_model)

  * lightest constructor used when transform sizes aren’t known at construct time; doesn’t initialize variables and responses, so this Model can’t be used to construct an Iterator; requires subsequent init_sizes() and init_map() calls.

- **~RecastModel ()**

  * destructor

- **void init_sizes** (const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset, short recast_resp_order)
update recast sizes and size Variables and Response members after alternate construction


  initialize recast indices and map callbacks after alternate construction

- void inverse_mats (void(*inv_vars_map)(const Variables &sub_model_vars, Variables &recast_vars), void(*inv_set_map)(const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set), void(*inv_pri_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast resp, Response &sub_model resp), void(*inv_sec_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast resp, Response &sub_model resp))

  provide optional inverse mappings

- void transform_vars (const Variables &recast_vars, Variables &sub_model_vars)

  perform transformation of Variables (recast –> sub-model)

- void transform_set (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)

  into sub_model_set for use with subModel.

- void transform_response (const Variables &recast_vars, const Variables &sub_model_vars, const Response &sub_model resp, Response &recast resp)

  perform transformation of Response (sub-model –> recast)

- void transform_response_map (const IntResponseMap &old_resp_map, IntResponseMap &new_resp_map)

  invoke transform_response() on each response within old_resp_map to create new_resp_map

- void inverse_transform_vars (const Variables &sub_model_vars, Variables &recast_vars)

  perform inverse transformation of Variables (sub-model –> recast)

- void inverse_transform_set (const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set)

  into sub_model_set for use with subModel.

- void inverse_transform_response (const Variables &sub_model_vars, const Variables &recast_vars, const Response &recast resp, Response &sub_model resp)

  perform inverse transformation of Response (recast –> sub-model)

- void submodel supports derivative estimation (bool sed_flag)

  override the submodel’s derivative estimation behavior

Protected Member Functions

- bool initialize_mapping (ParLevIter pl_iter)

  initialize model mapping, returns true if the variables size has changed

- bool finalize_mapping ()

  finalize model mapping, returns true if the variables size has changed

- void derived evaluate (const ActiveSet &set)

  portion of evaluate() specific to RecastModel (forward to subModel.evaluate())

- void derived evaluate_nowait (const ActiveSet &set)

  portion of evaluate_nowait() specific to RecastModel (forward to subModel.evaluate_nowait())
• const IntResponseMap & derived_synchronize ()
  
  portion of synchronize() specific to RecastModel (forward to subModel.synchronize())
• const IntResponseMap & derived_synchronize_nowait ()
  
  portion of synchronize_nowait() specific to RecastModel (forward to subModel.synchronize_nowait())
• Iterator & subordinate_iterator ()
  
  return sub-iterator, if present, within subModel
• Model & subordinate_model ()
  
  return subModel
• Model & surrogate_model ()
  
  return surrogate model, if present, within subModel
• Model & truth_model ()
  
  return truth model, if present, within subModel
• void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  
  add subModel to list and recurse into subModel
• void update_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())
  
  pass request to subModel if recursing and then update from it
• Interface & derived_interface ()
  
  return subModel interface
• size_t solution_levels () const
  
  return size of subModel::solnControlCostMap
• void solution_level_index (size_t lev_index)
  
  activate entry in subModel::solnControlCostMap
• size_t solution_level_index () const
  
  return active entry in subModel::solnControlCostMap
• RealVector solution_level_costs () const
  
  return cost estimates from subModel::solnControlCostMap
• Real solution_level_cost () const
  
  return active cost estimate from subModel::solnControlCostMap
• void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into subModel
• void surrogate_function_indices (const IntSet &surr_fn_indices)
  
  update the subModel’s surrogate response function indices (DataFitSurrModel::surrogateFnIndices)
• void surrogate_response_mode (short mode)
  
  update the subModel’s surrogate response mode (SurrogateModel::responseMode)
• const RealVector & error_estimates ()
  
  retrieve error estimates corresponding to the subModel
• void build_approximation (size_t index=NPOS)
  
  builds the subModel approximation
• bool build_approximation (const Variables &vars, const IntResponsePair &response_pr, size_t index=NPOS)
  
  builds the subModel approximation
• void update_approximation (bool rebuild_flag, size_t index=NPOS)
replaces data in the subModel approximation

- void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag, size_t index=NPOS)
  replaces data in the subModel approximation

- void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=NPOS)
  replaces data in the subModel approximation

- void append_approximation (bool rebuild_flag, size_t index=NPOS)
  appends data to the subModel approximation

- void append_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag, size_t index=NPOS)
  appends data to the subModel approximation

- void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag, size_t index=NPOS)
  appends data to the subModel approximation

- void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
  remove the previous data set addition to a surrogate (e.g., due to a previous append_approximation() call); flag manages storing of surrogate data for use in a subsequent push_approximation()

- void push_approximation ()
  push a previous approximation data state; reverse of pop_approximation

- bool push_available ()
  query for whether a trial increment is restorable within a surrogate

- void finalize_approximation ()
  finalize an approximation by applying all previous trial increments

- void store_approximation (size_t index=NPOS)
  move the current approximation into storage for later combination; the index of the stored set can be passed to allow replacement instead of augmentation (default is push_back)

- void restore_approximation (size_t index=NPOS)
  return an approximation from storage; the index identifies a particular stored data set (default is pop_back from stored)

- void remove_stored_approximation (size_t index=NPOS)
  remove a stored approximation, due to redundancy with the current approximation, prior to combination (default for no index is pop_back)

- void combine_approximation ()
  combine the current approximation with previously stored data sets

- void clear_stored ()
  clear stored approximations

- std::vector< Approximation > & approximations ()
  retrieve the set of Approximations from the subModel

- const RealVectorArray & approximation_coefficients (bool normalized=false)
  retrieve the approximation coefficients from the subModel

- void approximation_coefficients (const RealVectorArray &approx_coeffs, bool normalized=false)
  set the approximation coefficients within the subModel

- const RealVector & approximation_variances (const Variables &vars)
retrieve the approximation variances from the subModel

- const Pecos::SurrogateData & approximation_data (size_t index)
  retrieve the approximation data from the subModel

- void component_parallel_mode (short mode)
  RecastModel only supports parallelism in subModel, so this virtual function redefinition is simply a sanity check.

- size_t mi_parallel_level_index () const
  return subModel’s MI parallel level index

- short local_eval_synchronization ()
  return subModel local synchronization setting

- int local_eval_concurrency ()
  return subModel local evaluation concurrency

- bool derived_master_overload () const
  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to subModel)

- IntPtrIntPair estimate_partition_bounds (int max_eval_concurrency)
  estimate the minimum and maximum partition sizes that can be utilized by this Model

- void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set up RecastModel for parallel operations (request forwarded to subModel)

- void derived_init_serial ()
  set up RecastModel for serial operations (request forwarded to subModel).

- void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration within subModel

- void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the RecastModel (request forwarded to subModel)

- void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)
  Service subModel job requests received from the master. Completes when a termination message is received from stop_servers().

- void stop_servers ()
  executed by the master to terminate subModel server operations when RecastModel iteration is complete.

- void inactive_view (short view, bool recurse_flag=true)
  update the Model’s inactive view based on higher level (nested) context and optionally recurse into subModel

- const String & interface_id () const
  return the subModel interface identifier

- bool evaluation_cache (bool recurse_flag=true) const
  if recurse_flag, return the subModel evaluation cache usage

- bool restart_file (bool recurse_flag=true) const
  if recurse_flag, return the subModel restart file usage

- int derived_evaluation_id () const
  return the current evaluation id for the RecastModel

- void set_evaluation_reference ()
  set the evaluation counter reference points for the RecastModel (request forwarded to subModel)

- void fine_grained_evaluation_counters ()
  request fine-grained evaluation reporting within subModel
• void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  
  print the evaluation summary for the RecastModel (request forwarded to subModel)

• void warm_start_flag (const bool flag)
  set the warm start flag, including the orderedModels

• void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix

• bool db_lookup (const Variables &search_vars, const ActiveSet &search_set, Response &foundResp)
  RecastModel may need to map variables, asv before DB lookup, or responses after lookup.

• bool init_variables (const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr)
  initialize currentVariables and related info from the passed size/type info

• void init_response (size_t num_recast_primary_fns, size_t num_recast_secondary_fns, short recast_resp_order, bool reshape_vars)
  initialize currentResponse from the passed size info

• void reshape_response (size_t num_recast_primary_fns, size_t num_recast_secondary_fns)
  Reshape the RecastModel Response, assuming no change in variables or derivative information.

• void init_constraints (size_t num_recast_secondary_fns, size_t recast_secondary_offset, bool reshape_vars)
  initialize userDefinedConstraints from the passed size info

Protected Attributes

• Model subModel
  the sub-model underlying the transformations

• int recastModelEvalCntr
  local evaluation id counter used for id mapping

• IntActiveSetMap recastSetMap
  map of recast active set passed to derived_evaluate_nowait(). Needed for currentResponse update in synchronization routines.

• IntVariablesMap recastVarsMap
  map of recast variables used by derived_evaluate_nowait(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.

• IntVariablesMap subModelVarsMap
  map of subModel variables used by derived_evaluate_nowait(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.

• IntResponseMap recastResponseMap
  map of recast responses used by RecastModel::derived_synchronize() and RecastModel::derived_synchronize_nowait()

• IntIntMap recastIdMap
  mapping from subModel evaluation ids to RecastModel evaluation ids

Private Member Functions

• void initialize_data_from_submodel ()
  code shared among constructors to initialize base class data from submodel

• void update_from_model (Model &model)
  update current variables/labels/bounds/targets from subModel
13.167. RECASTMODEL CLASS REFERENCE

Private Attributes

- **Sizet2DArray varsMapIndices**
  For each subModel variable, identifies the indices of the recast variables used to define it (maps RecastModel variables to subModel variables; data is packed with only the variable indices employed rather than a sparsely filled N_{sm} x N_r matrix).

- **bool nonlinearVarsMapping**
  boolean set to true if the variables mapping involves a nonlinear transformation. Used in transform_set() to manage the requirement for gradients within the Hessian transformations. This does not require a BoolDeque for each individual variable, since response gradients and Hessians are managed per function, not per variable.

- **bool respMapping**
  set to true if non-NULL primaryRespMapping or secondaryRespMapping are supplied

- **Sizet2DArray primaryRespMapIndices**
  For each recast primary function, identifies the indices of the subModel functions used to define it (maps subModel response to RecastModel Response).

- **Sizet2DArray secondaryRespMapIndices**
  For each recast secondary function, identifies the indices of the subModel functions used to define it (maps subModel response to RecastModel response).

- **BoolDequeArray nonlinearRespMapping**
  array of BoolDeques, one for each recast response function. Each BoolDeque defines which subModel response functions contribute to the recast function using a nonlinear mapping. Used in transform_set() to augment the subModel function value/gradient requirements.

- **RealVector mappedError Estimates**
  mapping of subModel.error_estimates() through response mappings

- **void(* variablesMapping)(const Variables &recast_vars, Variables &sub_model_vars)**
  holds pointer for variables mapping function passed in ctor/initialize

- **void(* setMapping)(const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)**
  holds pointer for set mapping function passed in ctor/initialize

- **void(* primaryRespMapping)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)**
  holds pointer for primary response mapping function passed in ctor/initialize

  holds pointer for secondary response mapping function passed in ctor/initialize

- **void(* invVarsMapping)(const Variables &sub_model_vars, Variables &recast_vars)**
  holds pointer for optional inverse variables mapping function passed in inverse_mappings()”

- **void(* invSetMapping)(const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set)**
  holds pointer for optional inverse set mapping function passed in inverse_mappings()”

- **void(* invPriRespMapping)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_resp)**
  holds pointer for optional inverse primary response mapping function passed in inverse_mappings()”

- **void(* invSecRespMapping)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_resp)**
  holds pointer for optional inverse secondary response mapping function passed in inverse_mappings()”
Additional Inherited Members

13.167.1 Detailed Description

Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

The RecastModel class uses function pointers to allow recasting of the subModel input/output into new problem forms. For example, this is used to recast SBO approximate subproblems, multiobjective and least-squares reductions, and variable/response.

For now, making the assumption that variables mappings are ordered by submodel active continuous, discrete int, discrete string, discrete real variables, even though all current use cases are continuous only.

When not using the standard (full) constructor, client code must make sure to complete initialization before using the RecastModel’s mapping functions. Initialization steps:

1. sub model (all ctors do this)
2. init_sizes: once known, size Variables, Response, Constraints (full and intermediate ctor do this)
3. init_maps: set indices and callback pointers (only full ctor does this)

13.167.2 Constructor & Destructor Documentation

RecastModel ( const Model & sub_model, const Sizet2DArray & vars_map_indices, const SizetArray & vars_comps_totals, const BitArray & all_relax_di, const BitArray & all_relax_dr, bool nonlinear_vars_mapping, void(*)(const Variables &recast_vars, Variables &sub_model_vars) variables_map, void(*)(const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set) set_map, const Sizet2DArray & primary_res_map_indices, const Sizet2DArray & secondary_res_map_indices, size_t recast_secondary_offset, short recast_resp_order, const BoolDequeArray & nonlinear_resp_mapping, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) primary_resp_map, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) secondary_resp_map )

standard (full) constructor; assumes provided sizes and map functions are final and constructs all member data

Default recast model constructor. Requires full definition of the transformation; if any mappings are NULL, they are assumed to remain so in later initialization or updates. Parameter vars_comps_totals indicates the number of each type of variable \{4 types\} x \{3 domains\} in the recast variable space. Note: recast_secondary_offset is the start index for equality constraints, typically num nonlinear ineq constraints.

References Dakota::abort_handler(), Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Variables::cv(), RecastModel::init_constraints(), RecastModel::init_response(), RecastModel::init_variables(), RecastModel::initialize_data_from_submodel(), Model::modelType, RecastModel::nonlinearRespMapping, Response::num_functions(), Model::numDerivVars, Model::numFns, RecastModel::primaryRespMapIndices, RecastModel::primaryRespMapping, RecastModel::respMapping, RecastModel::secondaryRespMapIndices, RecastModel::secondaryRespMapping, RecastModel::subModel, Model::supportsEstimDerivs, and RecastModel::variablesMapping.

RecastModel ( const Model & sub_model, const SizetArray & vars_comps_totals, const BitArray & all_relax_di, const BitArray & all_relax_dr, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset, short recast_resp_order )

alternate constructor; uses provided sizes to construct Variables, Response and Constraints so Model can be passed to an Iterator; requires subsequent init_maps() call.
This alternate constructor defers initialization of the function pointers until a separate call to initialize(), and accepts the minimum information needed to construct currentVariables, currentResponse, and userDefinedConstraints. The resulting model is sufficiently complete for passing to an Iterator. Parameter vars_comps_totals indicates the number of each type of variable \(4 \times 3\) in the recast variable space. Note: recast_secondary_offset is the start index for equality constraints, typically num nonlinear ineq constraints.

References RecastModel::init_sizes(), RecastModel::initialize_data_from_submodel(), Model::modelType, and Model::supportsEstimDerivs.

### 13.167.3 Member Function Documentation

```cpp
void init_maps ( const Sizet2DArray & vars_map_indices, bool nonlinear_vars_mapping, void(*)(const Variables &recast_vars, Variables &sub_model_vars) variables_map, void(*)(const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set) set_map, const Sizet2DArray & primary_resp_map_indices, const Sizet2DArray & secondary_resp_map_indices, const BoolDequeArray & nonlinear_resp_mapping, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) primary_resp_map, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) secondary_resp_map )
```

initialize recast indices and map callbacks after alternate construction

This function is used for late initialization of the recasting functions. It is used in concert with the alternate constructor.


Referenced by NonDGlobalInterval::core_run(), NonDLocalInterval::core_run(), DataTransformModel::DataTransformModel(), RandomFieldModel::initialize_recast(), AdaptedBasisModel::initialize_recast(), ActiveSubspaceModel::initialize_recast(), EffGlobalMinimizer::minimize_surrogates_on_model(), NonDLocalReliability::mpp_search(), NonDGlocalReliability::optimize_gaussian_process(), ProbabilityTransformModel::ProbabilityTransformModel(), and ScalingModel::ScalingModel().

```cpp
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]
```

portion of evaluate() specific to RecastModel (forward to subModel.evaluate())

The RecastModel is evaluated by an Iterator for a recast problem formulation. Therefore, the currentVariables, incoming active set, and output currentResponse all correspond to the recast inputs/outputs.

Reimplemented from Model.

References Response::active_set(), Model::current_response(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Model::evaluate(), RecastModel::recastModelEvalCntr, RecastModel::respMapping, RecastModel::subModel, RecastModel::transform_response(), RecastModel::transform_set(), RecastModel::transform_variables(), and Response::update().

Referenced by AdaptedBasisModel::derived_evaluate(), DataTransformModel::derived_evaluate(), ActiveSubspaceModel::derived_evaluate(), and RandomFieldModel::derived_evaluate().

```cpp
void eval_tag_prefix ( const String & eval_id_str ) [inline], [protected], [virtual]
```

set the hierarchical eval ID tag prefix

RecastModel just forwards any tags to its subModel

Reimplemented from Model.

References Model::eval_tag_prefix(), and RecastModel::subModel.
void update_from_model ( Model & model ) [private]

update current variables/labels/bounds/targets from subModel

Update inactive values and labels in currentVariables and inactive bound constraints in userDefinedConstraints from variables and constraints data within subModel.

References
Variables::acv(), Model::acv(), Variables::adiv(), Model::adiv(), Variables::adrv(), Model::adrv(), Variables::adsv(), Model::adsv(), Model::aleatDistParams, Model::aleatory_distribution_parameters(), Constraints::all_continuous_lower_bound(), Constraints::all_continuous_upper_bound(), Constraints::all_continuous_values(), Constraints::all_discrete_int_lower_bound(), Constraints::all_discrete_int_upper_bound(), Constraints::all_discrete_int_values(), Constraints::all_nonlinear_targets(), Constraints::all_weights(), Constraints::bound(), Constraints::continuous(), Constraints::discrete(), Constraints::integer(), Constraints::label(), Variables::all_continuous_variable(), Variables::all_discrete_variable(), Variables::all_integer_variable(), Variables::all_nonlinear_variable(), Variables::all_real_variable(), Variables::all_string_variable(), Variables::all_variable(), Variables::continuous(), Variables::discrete(), Variables::integer(), Variables::integer_range(), Variables::label(), Variables::nonlinear(), Variables::real(), Variables::string(), Variables::variable()

• RecastModel.cpp
• RecastModel.hpp

The documentation for this class was generated from the following files:

- RecastModel.hpp
- RecastModel.cpp
13.168 ReducedBasis Class Reference

Public Member Functions

- **ReducedBasis ()**
  
  `default constructor`

- void **set_matrix** (const RealMatrix &)

- const RealMatrix & **get_matrix** ()

- void **center_matrix** ()
  
  `center the matrix by scaling each column by its means`

- void **update_svd** (bool center_matrix_by_col_means=true)
  
  `ensure that the factorization is current, centering if requested`

- bool **is_valid** () const

- const Real & **get_singular_values_sum** () const

- const Real & **get_eigen_values_sum** () const

- const RealVector & **get_column_means** ()

- const RealVector & **get_singular_values** () const

- RealVector **get_singular_values** (const TruncationCondition &) const

- const RealMatrix & **get_left_singular_vector** () const
  
  `the num_observations n x num_observations n orthogonal matrix U; the left singular vectors are the first min(n,p) columns`

- const RealMatrix & **get_right_singular_vector_transpose** () const
  
  `the num_responses p x num_responses p orthogonal matrix V'; the right singular vectors are the first min(n,p) rows of V' (columns of V)`

Private Attributes

- RealMatrix **matrix**

- RealMatrix **workingMatrix**

- RealMatrix **U_matrix**

- RealVector **S_values**

- RealMatrix **VT_matrix**

- RealVector **column_means**

- bool **col_means_computed**

- bool **is_centered**

- bool **is_valid_svd**

- Real **singular_values_sum**

- Real **eigen_values_sum**

- TruncationCondition * **truncation**

13.168.1 Detailed Description

The **ReducedBasis** class is used to ... (TODO - RWH)

Class to manage data-driven dimension reduction. The passed matrix with num_observations n rows and num_responses p columns contains realizations of a set of responses. The class optionally centers the matrix by the column means. Stores a singular value decomposition of the passed data matrix \(X = U \times S \times V'\), which can also be used for PCA, where we seek an eigendecomposition of the covariance: \(X' \times X = V \times D \times V^{-1} = V \times S^2 \times V'\)

The documentation for this class was generated from the following files:
13.169  RelaxedVarConstraints Class Reference

Derived class within the Constraints hierarchy which employs relaxation of discrete variables.

Inheritance diagram for RelaxedVarConstraints:

```
Constraints
    |  RelaxedVarConstraints
```

Public Member Functions

- **RelaxedVarConstraints** (const SharedVariablesData &svd)
  lightweight constructor

- **RelaxedVarConstraints** (const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  standard constructor

- **~RelaxedVarConstraints**()
  destructor

- **void write** (std::ostream &s) const
  write a variable constraints object to an std::ostream

- **void read** (std::istream &s)
  read a variable constraints object from an std::istream

Additional Inherited Members

13.169.1 Detailed Description

Derived class within the Constraints hierarchy which employs relaxation of discrete variables.

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The RelaxedVarConstraints derived class combines continuous and discrete domain types through integer relaxation. The branch and bound method uses this approach (see Variables-::get_variables(problem_db) for variables type selection; variables type is passed to the Constraints constructor in Model).

13.169.2 Constructor & Destructor Documentation

RelaxedVarConstraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )

standard constructor

In this class, a relaxed data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators which use this class include: BranchBndOptimizer.
13.170. **RELAXEDVARIABLES CLASS REFERENCE**

References SharedVariablesData::all_relaxed_discrete_int(), SharedVariablesData::all_relaxed_discrete_real(), Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Constraints::sharedVarsData, and SharedVariablesData::view().

The documentation for this class was generated from the following files:

- RelaxedVarConstraints.h
- RelaxedVarConstraints.cpp

### 13.170 RelaxedVariables Class Reference

Derived class within the Variables hierarchy which employs the relaxation of discrete variables.

Inheritance diagram for RelaxedVariables:

```
Variables
   RelaxedVariables
```

#### Public Member Functions

- **RelaxedVariables** (const ProblemDescDB &problem_db, const std::pair<short, short> &view)
  
  *standard constructor*

- **RelaxedVariables** (const SharedVariablesData &svd)

  *lightweight constructor*

- ∼RelaxedVariables()

  *destructor*

#### Protected Member Functions

- void **read** (std::istream &s)
  
  *read a variables object from an std::istream*

- void **write** (std::ostream &s, unsigned short vars_part=ALL_VARS) const

  *write a variables object to an std::ostream, e.g., the console, optionally specifying which partition (all/active/inactive)*

- void **write_aprepro** (std::ostream &s) const

  *write a variables object to an std::ostream in aprepro format, e.g., a parameters file*

- void **read_tabular** (std::istream &s, unsigned short vars_part=ALL_VARS)

- void **write_tabular** (std::ostream &s, unsigned short vars_part=ALL_VARS) const

  *write a variables object in tabular format to an std::ostream, optionally specifying which partition (all/active/inactive)*

- void **write_tabular_labels** (std::ostream &s, unsigned short vars_part=ALL_VARS) const

  *write the labels in input spec order to a std::ostream, optionally specifying which partition (all/active/inactive)*

- template<typename Reader>

  void **read_core** (std::istream &s, Reader read_handler, unsigned short vars_part)
Implementation of reading various formats using the specified read handler, accounting for reordering due to relaxation.

- template<typename Writer >
  void write_core ( std::ostream &s, Writer write_handler, unsigned short vars_part) const

Implementation of writing various formats using the specified write handler, accounting for reordering due to relaxation.

Additional Inherited Members

13.170.1 Detailed Description

Derived class within the Variables hierarchy which employs the relaxation of discrete variables.

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The RelaxedVariables derived class combines continuous and discrete domain types but separates design, uncertain, and state variable types. The branch and bound method uses this approach (see Variables::get_variables(problem_db)).

13.170.2 Constructor & Destructor Documentation

RelaxedVariables ( const ProblemDescDB & problem_db, const std::pair< short, short > & view )

standard constructor

In this class, a relaxed data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators/strategies which use this class include: BranchBndOptimizer. Extract fundamental variable types and labels and merge continuous and discrete domains to create aggregate arrays and views.

References SharedVariablesData::all_relaxed_discrete_int(), SharedVariablesData::all_relaxed_discrete_real(), Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteStringVars, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), ProblemDescDB::get_sa(), Variables::sharedVarsData, and SharedVariablesData::view().

13.170.3 Member Function Documentation

void read_tabular ( std::istream & s, unsigned short vars_part = ALL_VARS ) [protected], [virtual]

Presumes variables object is appropriately sized to receive data

Reimplemented from Variables.

References RelaxedVariables::read_core().

void read_core ( std::istream & s, Reader read_handler, unsigned short vars_part ) [protected]

Implementation of reading various formats using the specified read handler, accounting for reordering due to relaxation.

Reordering is required in all read/write cases that will be visible to the user since all derived vars classes should use the same ordering for clarity. Neutral file I/O, binary streams, and packed buffers do not need to reorder (so long as read/write are consistent) since this data is not intended for public consumption.

References SharedVariablesData::active_components_totals(), Variables::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Variables::all_discrete_string_variable_labels(), SharedVariablesData::all_relaxed_discrete_int(), SharedVariablesData::all_relaxed_discrete_real(), Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteString-
Container class for response functions and their derivatives. **Response** provides the enveloper base class.

Inheritance diagram for Response:

```
Response
 |   |
 |   |
```

### Public Member Functions

- **Response ()**
  
  *default constructor*

- **Response (short type, const Variables &vars, const ProblemDescDB &problem_db)**
  
  *standard constructor built from problem description database*

- **Response (const SharedResponseData &srd, const ActiveSet &set)**
  
  *alternate constructor that shares response data*

- **Response (short type, const ActiveSet &set)**
  
  *alternate constructor using limited data without sharing*

- **Response (const SharedResponseData &srd)**
  
  *alternate constructor using limited data (explicit disallows implicit type conversion)*

- **Response (const Response &response)**
  
  *copy constructor*

- **virtual ~Response ()**
  
  *destructor*

- **Response operator= (const Response &response)**
  
  *assignment operator*

- **const SharedResponseData & shared_data () const**
  
  *return sharedRespData*

- **SharedResponseData & shared_data ()**
  
  *return sharedRespData*

- **size_t num_functions () const**
  
  *return the number of response functions*
• const ActiveSet & active_set() const
  return the active set

• void active_set(const ActiveSet &set)
  set the active set

• const ShortArray & active_set_request_vector() const
  return the active set request vector

• void active_set_request_vector(const ShortArray &asrv)
  set the active set request vector and verify consistent number of response functions

• const SizetArray & active_set_derivative_vector() const
  return the active set derivative vector

• void active_set_derivative_vector(const SizetArray &asdv)
  set the active set derivative vector and reshape functionGradients/functionHessians if needed

• const Real & function_value(size_t i) const
  return a function value

• Real & function_value_view(size_t i)
  return a “view” of a function value for updating in place

• const RealVector & function_values() const
  return all function values

• RealVector function_values_view()
  return all function values as a view for updating in place

• RealVector function_values_view() const
  return all function values as a view for accessing the function values vector from a const response

• void function_value(const Real &function_val, size_t i)
  set a function value

• void function_values(const RealVector &function_vals)
  set all function values

• const Real * function_gradient(int i) const
  return the i-th function gradient as a const Real*

• RealVector function_gradient_view(int i)
  return the i-th function gradient as a SerialDenseVector view (shallow copy) for updating in place

• RealVector function_gradient_view(int i) const
  return the i-th function gradient as a SerialDenseVector view (shallow copy) for accessing a column vector from a const matrix

• RealVector function_gradient_copy(int i) const
  return the i-th function gradient as a SerialDenseVector Teuchos::Copy (deep copy)

• const RealMatrix & function_gradients() const
  return all function gradients

• RealMatrix function_gradients_view()
  return all function gradients as a view for updating in place

• RealMatrix function_gradients_view() const
  return all function gradients as a view for updating in place

• void function_gradient(const RealVector &function_grad, int i)
  set a function gradient
function_gradients

const function_hessian

const function_hessian_view

const function_hessian_view

const function_hessians

function_hessians_view

function_hessians_view

function_hessians

field_values_view

field_values_view

field_values

field_coords_view

field_coords_view

field_coords

field_gradients_view

field_gradients_view

field_lengths

field_lengths

num_coords_per_field

set_coord_values

set all function gradients

return the i-th function Hessian

return the i-th function Hessian as a Teuchos::View (shallow copy) for updating in place

return the i-th function Hessian as a Teuchos::View (shallow copy) for accessing the i-th matrix within a const matrix array

return all function Hessians

return all function Hessians as Teuchos::Views (shallow copies) for updating in place

return all function Hessians as Teuchos::Views (shallow copies) for accessing the i-th matrix within a const matrix array

return const field values

return a “view” of a field value for updating in place

set a field value

return a “view” of a field value’s coordinates

return a const “view” of a field value’s coordinates

set a field value’s coordinates

return a view of the gradients of each field element

return a view of the Hessians of each field element

const IntVector & field_lengths

set the field lengths from sharedRespData

set the field lengths within sharedRespData

return the num_coords_per_field from sharedRespData

set the coordinate values per field
• const RealMatrix & get_coord_values (const size_t i) const
  return the coordinate values per field
• const StringArray & function_labels () const
  return the fine-grained (unrolled) response function identifier strings from sharedRespData
• void function_labels (const StringArray &labels)
  set the fine-grained (unrolled) response function identifier strings within sharedRespData
• const StringArray & field_group_labels ()
  return the user-provided field group labels instead of the unrolled labels available through function_labels()
• void read (std::istream &s, const unsigned short format=FLEXIBLE-results)
  read a response object of specified format from an std::istream
• void write (std::ostream &s) const
  write a response object to an std::ostream
• void read_annotated (std::istream &s)
  read a response object in annotated format from an std::istream
• void write_annotated (std::ostream &s) const
  write a response object in annotated format to an std::ostream
• void read_tabular (std::istream &s)
  read responseRep::functionValues in tabular format from an std::istream
• void write_tabular (std::ostream &s) const
  write responseRep::functionValues in tabular format to an std::ostream
• void write_tabular_labels (std::ostream &s) const
  write the labels to a tabular data stream
• void read (MPIUnpackBuffer &s)
  read a response object from a packed MPI buffer
• void write (MPIPackBuffer &s) const
  write a response object to a packed MPI buffer
• Response copy (bool deep_srd=false) const
  return a deep response copy of the contained responseRep for use in history mechanisms (SharedResponseData uses a shallow copy by default)
• int data_size ()
  return the number of doubles active in response. Used for sizing double* response_data arrays passed into read_data and write_data.
• void read_data (double *response_data)
  read from an incoming double* array
• void write_data (double *response_data)
  write to an incoming double* array
• void overlay (const Response &response)
  add incoming response to functionValues/Gradients/Hessians
• void update (const Response &response)
  Used in place of operator= when only results data updates are desired (functionValues/functionGradients/function-Hessians are updated, ASV/labels/id's/etc. are not). Care is taken to allow different derivative array sizing between the two response objects.
• void update (const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrix-Array &source_fn_hessians, const ActiveSet &source_set)
Overloaded form which allows update from components of a response object. Care is taken to allow different derivative array sizing.

- `void update_partial(size_t start_index_target, size_t num_items, const Response &response, size_t start_index_source)`

  partial update of this response object from another response object. The response objects may have different numbers of response functions.

- `void update_partial(size_t start_index_target, size_t num_items, const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set, size_t start_index_source)`

  Overloaded form which allows partial update from components of a response object. The response objects may have different numbers of response functions.

- `void reshape(size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag)`

  reshapes response data arrays

- `void reset()`

  resets all response data to zero

- `void reset_inactive()`

  resets all inactive response data to zero

- `bool is_null() const`

  function to check responseRep (does this handle contain a body)

- `virtual void set_scalar_covariance(RealVector &scalars)`

  method to set the covariance matrix defined for ExperimentResponse

- `virtual const ExperimentCovariance & experiment_covariance() const`

  retrieve the ExperimentCovariance structure

- `virtual void set_full_covariance(std::vector<RealMatrix> &matrices, std::vector<RealVector> &diagonals, RealVector &scalars, IntVector matrix_map_indices, IntVector diagonal_map_indices, IntVector scalar_map_indices)`

  method to set the full covariance matrices for ExperimentResponse

- `virtual Real apply_covariance(const RealVector &residuals) const`

  method to compute the triple product $v' \cdot \text{inv}(C) \cdot v$.

- `virtual void apply_covariance_inv_sqrt(const RealVector &residuals, RealVector &weighted_residuals) const`

  method to compute $(v' \cdot \text{inv}(C) \cdot v)^{1/2}$, to compute weighted residual

- `virtual void apply_covariance_inv_sqrt(const RealMatrix &gradients, RealMatrix &weighted_gradients) const`

- `virtual void apply_covariance_inv_sqrt(const RealSymMatrixArray &hessians, RealSymMatrixArray &weighted_hessians) const`

- `virtual Real get_covariance_diagonal(RealVector &diagonal) const`

  covariance determinant for one experiment (default 1.0)

- `virtual Real log_covariance_determinant() const`

  log of covariance determinant for one experiment (default 0.0)
CHAPTER 13. CLASS DOCUMENTATION

Protected Member Functions

- **Response** (BaseConstructor, const Variables &vars, const ProblemDescDB &problem_db)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Response** (BaseConstructor, const SharedResponseData &srd, const ActiveSet &set)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Response** (BaseConstructor, const ActiveSet &set)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Response** (BaseConstructor, const SharedResponseData &srd)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **virtual void copy_rep (Response *source_rep)**
  Implementation of data copy for Response letters (specialized by some derived letter types); pulls base class data from source_rep into the this object.

Protected Attributes

- **SharedResponseData sharedRespData**
  reference-counted instance of shared response data: id’s, labels

- **RealVector functionValues**
  abstract set of response functions

- **RealMatrix functionGradients**
  first derivatives of the response functions

- **RealSymMatrixArray functionHessians**
  second derivatives of the response functions

- **IntRealMatrixMap fieldCoords**
  coordinates for the field values

- **ActiveSet responseActiveSet**
  copy of the ActiveSet used by the Model to generate a Response instance

Private Member Functions

- **template<class Archive , typename OrdinalType , typename ScalarType >**
  void **write_sdm_col** (Archive &ar, int col, const Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &sdm) const
  write a column of a SerialDenseMatrix

- **template<class Archive , typename OrdinalType , typename ScalarType >**
  void **read_sdm_col** (Archive &ar, int col, Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &sdm)
  read a column of a SerialDenseMatrix

- **template<class Archive >**
  void **load** (Archive &ar, const unsigned int version)
  read a Response from an archive<class Archive>
13.171. RESPONSE CLASS REFERENCE

- template<class Archive >
  void load_rep (Archive &ar, const unsigned int version)
  
  read a Response letter object from an archive

- template<class Archive >
  void save (Archive &ar, const unsigned int version) const
  
  write a Response to an archive

- template<class Archive >
  void save_rep (Archive &ar, const unsigned int version) const
  
  write a Response letter object to an archive

- BOOST_SERIALIZATION_SPLIT_MEMBER () Response *get_response(short type
  
  Used by standard envelope constructor to instantiate a new letter class.

- Response * get_response (const SharedResponseData &srd, const ActiveSet &set) const
  
  Used by alternate envelope constructor to instantiate a new letter class.

- Response * get_response (short type, const ActiveSet &set) const
  
  Used by alternate envelope constructor to instantiate a new letter class.

- Response * get_response (const SharedResponseData &srd) const
  
  Used by copy() to instantiate a new letter class.

- Response * get_response (short type) const
  
  Used by read functions to instantiate a new letter class.

- void read_annotated_rep (std::istream &s)
  
  read a letter object in annotated format from an std::istream

- void write_annotated_rep (std::ostream &s) const
  
  write a letter object in annotated format to an std::ostream

- void read_rep (MPIUnpackBuffer &s)
  
  read a letter object from a packed MPI buffer

- void write_rep (MPIPackBuffer &s) const
  
  write a letter object to a packed MPI buffer

- void shape_rep (const ActiveSet &set, bool initialize=true)
  
  resizes the representation's containers

- void reshape_rep (size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag)
  
  resizes the representation's containers

- void read_gradients (std::istream &s, const ShortArray &asv, std::ostringstream &error)
  
  Read gradients from a freeform stream. Insert error messages.

- void read_hessians (std::istream &s, const ShortArray &asv, std::ostringstream &error)
  
  Read Hessians from a freeform stream. Insert error messages.

- void read_labeled_fn_vals (std::istream &s, const ShortArray &asv, std::ostringstream &errors)
  
  Read function values from an annotated stream. Insert error messages.

- void read_flexible_fn_vals (std::istream &s, const ShortArray &asv, std::ostringstream &errors)
  
  Read function values from a stream in a "flexible" way – ignoring any labels. Insert error messages into errors stream.

- bool failure_reported (std::istream &s)
  
  Check for FAIL in stream.
Private Attributes

- const Variables & vars
- const Variables const
  ProblemDescDB &problem_db const
- Response * responseRep
  pointer to the body (handle-body idiom)
- int referenceCount
  number of handle objects sharing responseRep

Friends

- class boost::serialization::access
- bool operator==(const Response &resp1, const Response &resp2)
  equality operator
- bool operator!=(const Response &resp1, const Response &resp2)
  inequality operator

13.171.1 Detailed Description

Container class for response functions and their derivatives. Response provides the enveloper base class.

The Response class is a container class for an abstract set of functions (functionValues) and their first (function-Gradients) and second (functionHessians) derivatives. The functions may involve objective and constraint functions (optimization data set), least squares terms (parameter estimation data set), or generic response functions (uncertainty quantification data set). For memory efficiency, it employs the ”letter-envelope idiom” approach to reference counting and representation sharing (see Coplien ”Advanced C++”), for which the base Response class serves as the envelope and one of its derived classes serves as the letter.

13.171.2 Member Data Documentation

RealMatrix functionGradients [protected]

first derivatives of the response functions

the gradient vectors (plural) are column vectors in the matrix (singular) with (row, col) = (variable index, response fn index).

Referenced by Response::field_gradients_view(), Response::function_gradient(), Response::function_gradient_copy(), Response::function_gradient_view(), Response::function_gradients(), and Response::function_gradients_view().

The documentation for this class was generated from the following file:

- DakotaResponse.hpp

13.172 RestartWriter Class Reference

Public Member Functions

- RestartWriter ()
  optional default ctor allowing a non-outputting RestartWriter
- RestartWriter (const String &write_restart_filename)
typical ctor taking a filename; this class encapsulates the output stream

- RestartWriter (std::ostream &write_restart_stream)
  alternate ctor taking a stream, helpful for testing; assumes client manages the output stream
- const String & filename ()
  output filename for this writer
- void append_prp (const ParamResponsePair &prp_in)
  add the passed pair to the restart file
- void flush ()
  flush the restart stream so we have a complete restart record should Dakota abort

Private Member Functions

- RestartWriter (const RestartWriter &)
  copy constructor is disallowed due to file stream
- const RestartWriter & operator= (const RestartWriter &)
  assignment is disallowed due to file stream

Private Attributes

- String restartOutputFilename
  the name of the restart output file
- std::ofstream restartOutputFS
  Binary stream to which restart data is written.
- boost::scoped_ptr<br boost::archive::binary_oarchive > restartOutputArchive
  Binary output archive to which data is written (pointer since no default ctor for oarchive and may not be initialized);

13.172.1 Detailed Description

Component for writing restart files. Creation and destruction of archive and associated stream are managed here.

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp

13.173 ResultsDBAny Class Reference

Public Member Functions

- template<typename StoredType >
  void array_allocate (const StrStrSizet &iterator_id, const std::string &data_name, size_t array_size, const MetaDataType &metadata)
  allocate an entry with sized array of the StoredType, e.g., array across response functions or optimization results sets
**CHAPTER 13. CLASS DOCUMENTATION**

- template<typename StoredType>
  ```cpp
  void array_insert (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, const StoredType &sent_data) 
  insert sent_data in specified position in previously allocated array
  ```

- template<typename StoredType>
  ```cpp
  StoredType get_data (const StrStrSizet &iterator_id, const std::string &data_name) const 
  return requested data by value in StoredType
  ```

- template<typename StoredType>
  ```cpp
  StoredType get_array_data (const StrStrSizet &iterator_id, const std::string &data_name, size_t index) const 
  return requested data from array by value in StoredType
  ```

- template<typename StoredType>
  ```cpp
  const StoredType * get_data_ptr (const StrStrSizet &iterator_id, const std::string &result_key) const 
  return pointer to stored data entry
  ```

- template<typename StoredType>
  ```cpp
  const StoredType * get_array_data_ptr (const StrStrSizet &iterator_id, const std::string &data_name, size_t index) const 
  return pointer to stored data at given array location
  ```

- void insert (const StrStrSizet &iterator_id, const std::string &data_name, const boost::any &result, const MetaDataType &metadata) 
  record addition with metadata map

- void dump_data (std::ostream &output_stream) 
  coarsely dump the data to the passed output stream

- void print_data (std::ostream &output_stream) 
  pretty print the data to the passed output stream

**Private Member Functions**

- const ResultsValueType & lookup_data (const StrStrSizet &iterator_id, const std::string &data_name) const 
  attempt to find the requested data, erroring if not found

- template<typename StoredType>
  ```cpp
  StoredType cast_data (const boost::any &dataholder) const 
  cast the reference to any data to the requested type
  ```

- template<typename StoredType>
  ```cpp
  const StoredType * cast_data_ptr (const boost::any *dataholder) const 
  cast the pointer to any data to the requested type
  ```

- void print_metadata (std::ostream &os, const MetaDataType &md) 
  print metadata to ostream

- void extract_data (const boost::any &dataholder, std::ostream &os) 
  determine the type of contained data and output it to ostream

- void output_data (const std::vector< double > &data, std::ostream &os) 
  output data to ostream

- void output_data (const std::vector< RealVector > &data, std::ostream &os) 
  output data to ostream
13.173. RESULTSDBANY CLASS REFERENCE

- void output_data (const std::vector<std::string> &data, std::ostream &os)
  
  output data to ostream

- void output_data (const std::vector<std::vector<std::string>> &data, std::ostream &os)
  
  output data to ostream

- void output_data (const RealMatrix &data, std::ostream &os)
  
  output data to ostream

- void output_data (const RealMatrix &data, std::ostream &os)
  
  output data to ostream

Private Attributes

- std::map<ResultsKeyType, ResultsValueType> iteratorData
  
  core data storage (map from key to value type)

13.173.1 Detailed Description

Class: ResultsDBAny Description: A map-based container to store DAKOTA Iterator results in underlying boost::any, with optional metadata

13.173.2 Member Function Documentation

void array_insert ( const StrStrSizet & iterator_id, const std::string & data_name, size_t index, const StoredType & sent_data )

insert sent_data in specified position in previously allocated array

insert requires previous allocation, and does not allow metadata update

References Dakota::abort_handler(), ResultsDBAny::iteratorData, and Dakota::make_key().

void insert ( const StrStrSizet & iterator_id, const std::string & data_name, const boost::any & result, const MetaDataType & metadata )

record addition with metadata map

Add or update existing entry

References ResultsDBAny::iteratorData, and Dakota::make_key().

Referenced by ResultsDBAny::array_allocate().

void extract_data ( const boost::any & dataholder, std::ostream & os ) [private]

determine the type of contained data and output it to ostream

Extract the data from the held any and map to supported concrete types int double RealVector (Teuchos::SerialDenseVector<int,double>) RealMatrix (Teuchos::SerialDenseMatrix<int,double>)

References ResultsDBAny::output_data().

Referenced by ResultsDBAny::dump_data(), and ResultsDBAny::print_data().

The documentation for this class was generated from the following files:

- ResultsDBAny.hpp
- ResultsDBAny.cpp
13.174  **ResultsEntry**< StoredType >  Class Template Reference

Class to manage in-core vs. file database lookups.

**Public Member Functions**

- **ResultsEntry** (const ResultsManager &results_mgr, const StrStrSizet &iterator_id, const std::string &data_name)
  
  *Construct ResultsEntry containing retrieved item of StoredType.*

- **ResultsEntry** (const ResultsManager &results_mgr, const StrStrSizet &iterator_id, const std::string &data_name, size_t array_index)
  
  *Construct ResultsEntry to retrieve item array_index from array of StoredType.*

**Private Member Functions**

- **ResultsEntry** ()

  *return a reference to the stored data, whether from core or file*

**Private Attributes**

- bool coreActive

  *whether the ResultsManager has an active in-core database*

- StoredType dbData

  *data retrieved from file data base*

- const StoredType *dbDataPtr

  *non-const pointer to const data we don’t own in the core case*

13.174.1  **Detailed Description**

template<typename StoredType> class Dakota::ResultsEntry< StoredType >

Class to manage in-core vs. file database lookups.

ResultsEntry manages database lookups. If a core database is available, will return a reference directly to the
stored data; if disk, will return reference to a local copy contained in this class. Allows disk-stored data to persist
for minimum time during lookup to support true out-of-core use cases.

13.174.2  **Constructor & Destructor Documentation**

ResultsEntry ( ) [private]

return a reference to the stored data, whether from core or file

*default construction disallowed: data must be initialized from DB lookup if needed*

The documentation for this class was generated from the following file:

- ResultsManager.hpp
13.175 ResultsFileError Class Reference

exception throw for other results file read error

Inheritance diagram for ResultsFileError:

```
FileReadException
    |
    v
ResultsFileError
```

Public Member Functions

- **ResultsFileError** (const std::string &msg)

13.175.1 Detailed Description

exception throw for other results file read error

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp

13.176 ResultsID Class Reference

Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each Iterator::run() call creates or increments this count for its string identifier.

Public Member Functions

- **size_t increment_id** (const std::string &methodName, const std::string &methodID)
  
  explicitly increment the iterator results ID, init to 1 if needed

- **size_t get_id** (const std::string &methodName, const std::string &methodID)

  get (possibly creating) a unique iterator results ID for the passed name

- **size_t get_id** (const std::string &methodName, const std::string &methodID) const

  get a unique iterator results ID for the passed name (const version errors if not found)

Static Public Member Functions

- static ResultsID & instance ()

  get the single unique instance of ResultsID

Private Member Functions

- **ResultsID ()**

  Private constructor for ResultsID.

- **~ResultsID ()**

  Private destructor for ResultsID.
• **ResultsID** (ResultsID const &)
  
  Private copy constructor for ResultsID.

• **ResultsID & operator=** (ResultsID const &)
  
  Private assignment operator for ResultsID.

### Private Attributes

- std::map<std::pair<std::string, std::string>, size_t> idMap
  
  Storage for the results IDs

13.176.1 Detailed Description

Get a globally unique 1-based execution number for a given iterator name (combination of methodName and methodID) for use in results DB. Each Iterator::run() call creates or increments this count for its string identifier.

The documentation for this class was generated from the following files:

- ResultsManager.hpp
- ResultsManager.cpp

### 13.177 ResultsManager Class Reference

Results manager for iterator final data.

**Public Member Functions**

- **ResultsManager ()**
  
  Default constructor: no databases active until initialize called

- void **initialize** (const std::string &base_filename)
  
  Initialize the results manager to manage an in-core database, writing to the specified file name

- bool **active ()** const
  
  Whether any databases are active

- void **write_databases ()**
  
  Write in-core databases to file.

- template<typename StoredType >
  
  void **insert** (const StrStrSizet &iterator_id, const std::string &data_name, const StoredType &sent_data, const MetaDataType metadata=MetaDataType())
  
  Insert data

- void **insert** (const StrStrSizet &iterator_id, const std::string &data_name, StringMultiArrayConstView sma_labels, const MetaDataType metadata=MetaDataType())

- template<typename StoredType >
  
  void **array_allocate** (const StrStrSizet &iterator_id, const std::string &data_name, size_t array_size, const MetaDataType metadata=MetaDataType())
  
  Allocate an entry with array of StoredType of array_size for future insertion; likely move to non-templated accessors for these
template<typename StoredType>
void array_insert (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, const StoredType &sent_data)

insert into a previously allocated array of StoredType at index specified; metadata must be specified at allocation

template<typename StoredType>
void array_insert (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, StringMultiArrayConstView sent_data)

specialization: insert a SMACV into a previously allocated array of StringArrayStoredType at index specified; metadata must be specified at allocation

Public Attributes

- **ResultsNames results_names**
  Copy of valid results names for when manager is passed around.

Private Member Functions

- template<typename StoredType>
  StoredType core_lookup (const StrStrSizet &iterator_id, const std::string &data_name) const
  retrieve in-core entry given by id and name

- template<typename StoredType>
  StoredType * core_lookup_ptr (const StrStrSizet &iterator_id, const std::string &data_name) const
  retrieve data via pointer to avoid copy; work-around for Boost any use of pointer (could use utilib::Any)

- template<typename StoredType>
  StoredType core_lookup (const StrStrSizet &iterator_id, const std::string &data_name, size_t index) const
  retrieve data from in-core array of StoredType at given index

- template<typename StoredType>
  const StoredType * core_lookup_ptr (const StrStrSizet &iterator_id, const std::string &data_name, size_t index) const
  retrieve data via pointer to entry in in-core array

- template<typename StoredType>
  void file_lookup (StoredType &db_data, const StrStrSizet &iterator_id, const std::string &data_name) const
  retrieve requested data into provided db_data StoredType

Private Attributes

- bool coreDBActive
  whether the in-core database is active

- std::string coreDBFilename
  filename for the in-core database

- bool hdf5DBActive
  whether the file database is active

- boost::scoped_ptr< ResultsDBAny > coreDB
  In-core database, with option to flush to file at end.

- boost::shared_ptr< ResultsDBHDF5 > hdf5DB
  File-based database; using shared_ptr due to potentially incomplete type and requirements for checked_delete in debug builds.
Friends

- template<typename StoredType >
  class ResultsEntry
  
  *ResultsEntry is a friend of ResultsManager.*

### 13.177.1 Detailed Description

Results manager for iterator final data.

The results manager provides the API for posting and retrieving iterator results data (and eventually run config/statistics). It can manage a set of underlying results databases, in or out of core, depending on configuration.

The key for a results entry is documented in results_types.hpp, e.g., tuple< std::string, std::string, size_t, std::string >

For now, using concrete types for most insertion, since underlying databases like HDF5 might need concrete types; though template parameter for array allocation and retrieval.

All insertions overwrite any previous data.

The documentation for this class was generated from the following files:

- ResultsManager.hpp
- ResultsManager.cpp

### 13.178 ResultsNames Class Reference

List of valid names for iterator results.

**Public Member Functions**

- ResultsNames ()

  *Default constructor initializes all valid names.*

**Public Attributes**

- size_t namesVersion
- std::string best_cv
- std::string best_div
- std::string best_dsv
- std::string bestDrv
- std::string best_fns
- std::string moments_std
- std::string moments_central
- std::string moments_std_num
- std::string moments_central_num
- std::string moments_std_exp
- std::string moments_central_exp
- std::string moments_cis
- std::string extreme_values
- std::string map_resp_prob
- std::string map_resp_rel
13.179. RICHEXTRAPVERIFICATION CLASS REFERENCE

- std::string map_resp_genrel
- std::string map_prob Resp
- std::string map_rel Resp
- std::string map_genrel Resp
- std::string pdf_histograms
- std::string correl_simple_all
- std::string correl_simple_io
- std::string correl_partial_io
- std::string correl_simple_rank_all
- std::string correl_simple_rank_io
- std::string correl_partial_rank_io
- std::string pce_coeffs
- std::string pce_coeff_labels
- std::string cv_labels
- std::string div_labels
- std::string dsv_labels
- std::string drv_labels
- std::string fn_labels

13.178.1 Detailed Description

List of valid names for iterator results.

All data in the ResultsNames class is public, basically just a struct.

The documentation for this class was generated from the following file:

- ResultsManager.hpp

13.179 RichExtrapVerification Class Reference

Class for Richardson extrapolation for code and solution verification.

Inheritance diagram for RichExtrapVerification:

```
   ___________
  |          |
  |    Iter  |
  |__________|
       |
    Analyzer
       |
  ___________
 |     Verif |
 |___________|
         |
 RichExtrapVerification
```
Public Member Functions

- **RichExtrapVerification** (ProblemDescDB &problem_db, Model &model)
  
  *constructor*

- ~**RichExtrapVerification** ()
  
  *destructor*

- void **core_run** ()
  
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- void **print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  
  print the final iterator results

Private Member Functions

- void **estimate_order** ()
  
  perform a single estimation of convOrder using extrapolation()

- void **converge_order** ()
  
  iterate using extrapolation() until convOrder stabilizes

- void **converge_qoi** ()
  
  iterate using extrapolation() until QOIs stabilize

- void **extrapolation** (const RealVector &refine_triple, RealMatrix &qoi_triples)
  
  estimate convOrder from refinement and quantity of interest (QOI) triples

- void **extrapolate_result** (const RealVector &refine_triple, const RealMatrix &qoi_triples)
  
  predict the converged value based on the convergence rate and the value of Phi

Private Attributes

- unsigned short **studyType**
  
  internal code for extrapolation study type: SUBMETHOD_{CONVERGE_ORDER, CONVERGE_QOI, ESTIMATE_ORDER}

- size_t **numFactors**
  
  number of refinement factors defined from active state variables

- RealVector **initialCVars**
  
  initial reference values for refinement factors

- size_t **factorIndex**
  
  the index of the active factor

- Real **refinementRate**
  
  rate of mesh refinement (default = 2.)

- RealMatrix **convOrder**
  
  the orders of convergence of the QOIs (numFunctions by numFactors)

- RealMatrix **extrapQOI**
  
  the extrapolated value of the QOI (numFunctions by numFactors)

- RealMatrix **numErrorQOI**
  
  the numerical uncertainty associated with level of refinement (numFunctions by numFactors)

- RealVector **refinementRefPt**
  
  This is a reference point reported for the converged extrapQOI and numErrorQOI. It currently corresponds to the coarsest mesh in the final refinement triple.
Additional Inherited Members

13.179.1 Detailed Description

Class for Richardson extrapolation for code and solution verification.

The RichExtrapVerification class contains several algorithms for performing Richardson extrapolation.

13.179.2 Member Function Documentation

void core_run() [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Dakota::abort_handler(), Model::continuous_variables(), RichExtrapVerification::converge_order(), RichExtrapVerification::converge_qoi(), RichExtrapVerification::convOrder, RichExtrapVerification::estimate_order(), RichExtrapVerification::extrapQOI, RichExtrapVerification::initialCVars, Iterator::iteratedModel, RichExtrapVerification::numErrorQOI, RichExtrapVerification::numFactors, Analyzer::numFunctions, Iterator::outputLevel, RichExtrapVerification::refinementRefPt, and RichExtrapVerification::studyType.

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Verification.

References Model::continuous_variable_labels(), RichExtrapVerification::convOrder, Dakota::copy_data(), RichExtrapVerification::extrapQOI, Iterator::iteratedModel, RichExtrapVerification::numErrorQOI, Verification::print_results(), RichExtrapVerification::refinementRate, RichExtrapVerification::refinementRefPt, and Model::response_labels().

void estimate_order() [private]

perform a single estimation of convOrder using extrapolation()

This algorithm executes a single refinement triple and returns convergence order estimates.


Referenced by RichExtrapVerification::core_run().

void converge_order() [private]

iterate using extrapolation() until convOrder stabilizes

This algorithm continues to refine until the convergence order estimate converges.


Referenced by RichExtrapVerification::core_run().
void converge_qoi( ) [private]
iterate using extrapolation() until QOIs stabilize
This algorithm continues to refine until the discretization error lies within a prescribed tolerance.
Referenced by RichExtrapVerification::core_run().
The documentation for this class was generated from the following files:
- RichExtrapVerification.hpp
- RichExtrapVerification.cpp

13.180 ScalingModel Class Reference
Scaling specialization of RecastModel.
Inheritance diagram for ScalingModel:

![Inheritance Diagram]

Public Member Functions
- **ScalingModel** (Model &sub_model)
  standard constructor
- **~ScalingModel** ()
  destructor
- RealVector cv_scaled2native (const RealVector &scaled_cv) const

Public members for help in final results recovery
- void resp_scaled2native (const Variables &native_vars, Response &updated_resp) const
  map responses from scaled to native space, updating provided Response in-place (on entry it’s scaled response, on exit it’s native)
- void secondary_resp_scaled2native (const RealVector &scaled_nln_cons, const ShortArray &asv, RealVector &native_fns) const
  Use scaled nonlinear constraints (sized total functions) to update the nonlinear constraints portion of the passed native_fns array.
- void response_modify_s2n (const Variables &native_vars, const Response &scaled_response, Response &native_response, int start_offset, int num_responses, bool response_unscale=true) const
  map responses from scaled to native space
Protected Member Functions

- void initialize_scaling (Model &sub_model)
  
  initialize scaling types, multipliers, and offsets; perform error checking

- void compute_scaling (int object_type, int auto_type, int num_vars, RealVector &lbs, RealVector &ubs, RealVector &targets, const StringArray &scale_strings, const RealVector &scales, IntArray &scale_types, RealVector &scale_mults, RealVector &scale_offsets)
  
  general helper function for initializing scaling types and factors on a vector of variables, functions, constraints, etc.

- RealMatrix lin_coeffs_modify_n2s (const RealMatrix &native_coeffs, const RealVector &cv_multipliers, const RealVector &lin_multipliers) const
  
  general linear coefficients mapping from native to scaled space

- bool compute_scale_factor (const Real lower_bound, const Real upper_bound, Real *multiplier, Real *offset)
  
  automatically compute a single scaling factor – bounds case

- bool compute_scale_factor (const Real target, Real *multiplier)
  
  automatically compute a single scaling factor – target case

- void print_scaling (const String &info, const IntArray &scale_types, const RealVector &scale_mults, const RealVector &scale_offsets, const StringArray &labels)
  
  print scaling information for a particular response type in tabular form

- bool need_resp_trans_byvars (const ShortArray &asv, int start_index, int num_resp) const
  
  determine if response transformation is needed due to variable transformations

- RealVector modify_n2s (const RealVector &native_vars, const IntArray &scale_types, const RealVector &multipliers, const RealVector &offsets) const
  
  general RealVector mapping from native to scaled variables vectors:

- RealVector modify_s2n (const RealVector &scaled_vars, const IntArray &scale_types, const RealVector &multipliers, const RealVector &offsets) const
  
  general RealVector mapping from scaled to native variables (and values)

- void response_modify_n2s (const Variables &scaled_vars, const Response &native_response, Response &iterator_response)
  
  map responses from native to scaled variable space

Static Protected Member Functions

- static short response_order (const Model &sub_model)
  
  helper to compute the recast response order during member initialization

- static bool scaling_active (const StringArray &scale_types)
  
  check whether the passed scale strings include any active (!= none) scale types

- static void variables_scaler (const Variables &native_vars, Variables &scaled_vars)
  
  RecastModel callback for variables scaling: transform variables from scaled to native (user) space.

- static void variables_unscaler (const Variables &native_vars, Variables &scaled_vars)
  
  RecastModel callback for inverse variables scaling: transform variables from native (user) to scaled space.

- static void primary_resp_scaler (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &iterator_response)
  
  RecastModel callback for primary response scaling: transform responses (grads, Hessians) from native (user) to scaled space.
- static void secondary_resp_scaler (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &scaled_response)

  RecastModel callback for secondary response scaling: transform constraints (grads, Hessians) from native (user) to scaled space.

Protected Attributes

- bool varsScaleFlag
  flag for variables scaling
- bool primaryRespScaleFlag
  flag for primary response scaling
- bool secondaryRespScaleFlag
  flag for secondary response scaling
- IntArray cvScaleTypes
  scale flags for continuous vars.
- RealVector cvScaleMultipliers
  scales for continuous variables
- RealVector cvScaleOffsets
  offsets for continuous variables
- IntArray responseScaleTypes
  scale flags for all responses
- RealVector responseScaleMultipliers
  scales for all responses
- RealVector responseScaleOffsets
  offsets for all responses (zero < for functions, not for nonlin con)
- IntArray linearIneqScaleTypes
  scale flags for linear ineq
- RealVector linearIneqScaleMultipliers
  scales for linear ineq constrs.
- RealVector linearIneqScaleOffsets
  offsets for linear ineq constrs.
- IntArray linearEqScaleTypes
  scale flags for linear eq.
- RealVector linearEqScaleMultipliers
  scales for linear constraints
- RealVector linearEqScaleOffsets
  offsets for linear constraints

Static Protected Attributes

- static ScalingModel * scaleModelInstance
  static pointer to this class for use in static callbacks
Additional Inherited Members

13.180.1 Detailed Description

Scaling specialization of RecastModel.

Specialization of RecastModel to scale Variables and/or Responses
This class provides a simple constructor that forwards to the more complicated RecastModel API.

13.180.2 Constructor & Destructor Documentation

ScalingModel ( Model & sub_model )

standard constructor

This constructor computes various indices and mappings, then updates the properties of the RecastModel.

References Model::cv(), ScalingModel::cvScaleTypes, Model::div(), Model::drv(), Model::dsv(), RecastModel::init_maps(), ScalingModel::initialize_scaling(), RecastModel::inverse_mappings(), Model::num_functions(), Model::num_primary_fns(), Model::outputLevel, ScalingModel::primary_resp_scaler(), Model::primary_response_fn_weights(), ScalingModel::primaryRespScaleFlag, ScalingModel::responseScaleTypes, ScalingModel::scaleModelInstance, ScalingModel::secondary_resp_scaler(), ScalingModel::secondaryRespScaleFlag, ScalingModel::variables_scaler(), ScalingModel::variables_unscaler(), and ScalingModel::varsScaleFlag.

13.180.3 Member Function Documentation

RealVector cv_scaled2native ( const RealVector & scaled_cv ) const

Public members for help in final results recovery

recover native variable values from the scaled space

Since this convenience function is public, it must have a fall-through to return a copy for when this scaling type isn’t active.

References ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ScalingModel::modify_s2n(), and ScalingModel::varsScaleFlag.

Referenced by LeastSq::post_run(), and Optimizer::post_run().

void resp_scaled2native ( const Variables & native_vars, Response & updated_resp ) const

map responses from scaled to native space, updating provided Response in-place (on entry it’s scaled response, on exit it’s native)

Since this convenience function is public, it must behave correctly when this scale type isn’t active. It does, because it modifies in-place

References Response::active_set_request_vector(), Response::copy(), ScalingModel::needResp_trans_byvars(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), ScalingModel::primaryRespScaleFlag, ScalingModel::response_modify_s2n(), ScalingModel::secondaryRespScaleFlag, and Response::update_partial().

Referenced by Optimizer::post_run().

void secondaryResp_scaled2native ( const RealVector & scaled_nln_cons, const ShortArray & asv, RealVector & native_fns ) const

Use scaled nonlinear constraints (sized total functions) to update the nonlinear constraints portion of the passed native_fns array.
Since this convenience function is public, it must have a fall-through to return a copy for when this scaling type isn’t active.

References Dakota::copy_data_partial(), ScalingModel::modify_s2n(), ScalingModel::need_resps_trans_byvars(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), ScalingModel::responseScaleMultipliers, ScalingModel::responseScaleOffsets, ScalingModel::responseScaleTypes, and ScalingModel::secondaryRespScaleFlag.

```cpp
void response_modify_s2n ( const Variables & native_vars, const Response & scaled_response, Response & native_response, int start_offset, int num_responses, bool unscale_resp = true ) const
```

map responses from scaled to native space

Unscaling response mapping: modifies response from scaled (iterator) to native (user) space. Maps num_responses starting at response_offset. If response_unscale = false, only variables will be unscaled, and responses left in scaled space.

References Response::active_set(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::all_continuous_variables(), Variables::continuous_variable_ids(), Variables::continuous_variables(), Dakota::copy_data(), Variables::cv(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_labels(), Response::function_values(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Model::num_primary_fns(), Model::outputLevel, ActiveSet::request_vector(), ScalingModel::responseScaleMultipliers, ScalingModel::responseScaleOffsets, ScalingModel::responseScaleTypes, and Dakota::write_precision.

Referenced by LeastSq::get_confidence_intervals(), and ScalingModel::resp_scaled2native().

```cpp
void initialize_scaling ( Model & sub_model ) [protected]
```

initialize scaling types, multipliers, and offsets; perform error checking

Initialize scaling types, multipliers, and offsets. Update the iteratedModel appropriately

References Dakota::abort_handler(), ScalingModel::compute_scaling(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variable_labels(), Model::continuous_variables(), Dakota::copy_data(), Model::cv(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ScalingModel::lin_coeffs_modify_n2s(), Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), ScalingModel::linearEqScaleMultipliers, ScalingModel::linearEqScaleOffsets, ScalingModel::linearEqScaleTypes, ScalingModel::linearIneqScaleMultipliers, ScalingModel::linearIneqScaleOffsets, ScalingModel::linearIneqScaleTypes, ScalingModel::modify_n2s(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_functions(), Model::num_linear_eq_constraints(), Model::num_linear_ineq_constraints(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), Model::outputLevel, ScalingModel::primaryRespScaleFlag, ScalingModel::print_scaling(), Model::response_labels(), ScalingModel::responseScaleMultipliers, ScalingModel::responseScaleOffsets, ScalingModel::responseScaleTypes, ScalingModel::scaling_active(), Model::scalingOpts, ScalingModel::secondaryRespScaleFlag, Model::supports_derivative_estimation(), Model::supports_EstimDerivs, and ScalingModel::varsScaleFlag.

Referenced by ScalingModel::ScalingModel().

```cpp
RealMatrix lin_coeffs_modify_n2s ( const RealMatrix & src_coeffs, const RealVector & cv_multipliers, const RealVector & lin_multipliers ) const [protected]
```

general linear coefficients mapping from native to scaled space
compute scaled linear constraint matrix given design variable multipliers and linear scaling multipliers. Only scales components corresponding to continuous variables so for src_coeffs of size MxN, lin_multipliers.size() ≤ M, cv_multipliers.size() ≤ N
Referenced by ScalingModel::initialize_scaling().

```cpp
void variables_scaler ( const Variables & scaled_vars, Variables & native_vars ) [static], [protected]
RecastModel callback for variables scaling: transform variables from scaled to native (user) space.
Variables map from iterator/scaled space to user/native space using a RecastModel.
References Variables::continuous_variable_labels(), Variables::continuous_variables(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ScalingModel::modify_s2n(), Model::outputLevel, and ScalingModel::scaleModelInstance.
Referenced by ScalingModel::ScalingModel().
```

```cpp
void secondary_resp_scaler ( const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response ) [static], [protected]
RecastModel callback for secondary response scaling: transform constraints (grads, Hessians) from native (user) to scaled space.
Constraint function map from user/native space to iterator/scaled/combined space using a RecastModel.
References Response::active_set_request_vector(), ScalingModel::need_resp_trans_byvars(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), Model::outputLevel, ScalingModel::response_modify_n2s(), ScalingModel::scaleModelInstance, ScalingModel::secondaryRespScaleFlag, and Response::update_partial().
Referenced by ScalingModel::ScalingModel().
```

```cpp
bool need_resp_trans_byvars ( const ShortArray & asv, int start_index, int num_resp ) const
[protected]
determine if response transformation is needed due to variable transformations
Determine if variable transformations present and derivatives requested, which implies a response transformation is necessary
References ScalingModel::varsScaleFlag.
Referenced by ScalingModel::primary_resp_scaler(), ScalingModel::resp_scaled2native(), ScalingModel::secondary_resp_scaled2native(), and ScalingModel::secondary_resp_scaler().
```

```cpp
RealVector modify_n2s ( const RealVector & native_vars, const IntArray & scale_types, const RealVector & multipliers, const RealVector & offsets ) const [protected]
general RealVector mapping from native to scaled variables vectors:
general RealVector mapping from native to scaled variables; loosely, in greatest generality: scaled_var = log((native_var - offset) / multiplier )
Referenced by ScalingModel::initialize_scaling(), and ScalingModel::variables_unscaler().
```

```cpp
RealVector modify_s2n ( const RealVector & scaled_vars, const IntArray & scale_types, const RealVector & multipliers, const RealVector & offsets ) const [protected]
general RealVector mapping from scaled to native variables (and values)
general RealVector mapping from scaled to native variables and/or vals; loosely, in greatest generality: scaled_var = (LOG_BASE^ scaled_var) * multiplier + offset
```
Referenced by ScalingModel::cv_scaled2native(), ScalingModel::secondary_resp_scaled2native(), and ScalingModel::variables_scaler().

```cpp
void response_modify_n2s ( const Variables & native_vars, const Response & native_response, Response & recast_response, int start_offset, int num_responses ) const [protected]
```

map responses from native to scaled variable space

Scaling response mapping: modifies response from a model (user/native) for use in iterators (scaled). Maps num_responses starting at response_offset

References Response::active_set(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::all_continuous_variables(), Variables::all_continuous_variable_ids(), Variables::all_continuous_variables(), Dakota::copy_data(), Variables::cv(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_value(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Model::num_primary_fns(), Model::outputLevel, ActiveSet::request_vector(), ScalingModel::responseScaleMultipliers, ScalingModel::responseScaleOffsets, ScalingModel::responseScaleTypes, and Dakota::write_precision.

Referenced by ScalingModel::primary_resp_scaler(), and ScalingModel::secondary_resp_scaler().

### 13.180.4 Member Data Documentation

**ScalingModel * scaleModelInstance [static], [protected]**

static pointer to this class for use in static callbacks

initialization of static needed by **RecastModel**

Referenced by ScalingModel::primary_resp_scaler(), ScalingModel::ScalingModel(), ScalingModel::secondary_resp_scaler(), ScalingModel::variables_scaler(), and ScalingModel::variables_unscaler().

The documentation for this class was generated from the following files:

- ScalingModel.hpp
- ScalingModel.cpp

### 13.181 ScalingOptions Class Reference

Simple container for user-provided scaling data, possibly expanded by replicates through the models.

#### Public Member Functions

- **ScalingOptions ()**
  
  default ctor: no scaling specified

- **ScalingOptions (const StringArray & cv_st, RealVector cv_s, const StringArray &pri_st, RealVector pri_s, const StringArray &nln_ineq_st, RealVector nln_ineq_s, const StringArray &nln_eq_st, RealVector nln_eq_s, const StringArray &lin_ineq_st, RealVector lin_ineq_s, const StringArray &lin_eq_st, RealVector lin_eq_s)**

  standard ctor: scaling from problem DB
Public Attributes

- `StringArray cvScaleTypes`
- `RealVector cvScales`
- `StringArray priScaleTypes`
- `RealVector priScales`
- `StringArray nlnIneqScaleTypes`
- `RealVector nlnIneqScales`
- `StringArray nlnEqScaleTypes`
- `RealVector nlnEqScales`
- `StringArray linIneqScaleTypes`
- `RealVector linIneqScales`
- `StringArray linEqScaleTypes`
- `RealVector linEqScales`

13.181.1 Detailed Description

Simple container for user-provided scaling data, possibly expanded by replicates through the models.

The documentation for this class was generated from the following file:

- DakotaModel.hpp

13.182 ScilabInterface Class Reference

Inheritance diagram for ScilabInterface:

```
    Interface
     |                |
     |                |
     v                v
ApplicationInterface
     |                |
     |                |
     v                v
DirectApplicInterface
     |                |
     |                |
     v                v
ScilabInterface
```

Public Member Functions

- `ScilabInterface (const ProblemDescDB &problem_db)`
  
  Constructor: start Matlab engine.

- `~ScilabInterface ()`
  
  Destructor: close Matlab engine.
Protected Member Functions

- virtual int derived_map_ac (const String &ac_name)
  execute an analysis code portion of a direct evaluation invocation
- int scilab_engine_run (const String &ac_name)
  principal Scilab execute function

Protected Attributes

- int scilabEngine
  identifier for the running Scilab engine

13.182.1 Detailed Description

Specialization of DirectApplicInterface to link to Scilab analysis drivers. Includes convenience functions to map data to/from Scilab.

The documentation for this class was generated from the following files:

- ScilabInterface.hpp
- ScilabInterface.cpp

13.183 SensAnalysisGlobal Class Reference

Class for a utility class containing correlation calculations and variance-based decomposition.

Public Member Functions

- SensAnalysisGlobal ()
  constructor
- ~SensAnalysisGlobal ()
  destructor
- void compute_correlations (const VariablesArray &vars_samples, const IntResponseMap &resp_samples, const StringSetArray &dss_vals)
  computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank
- void compute_correlations (const RealMatrix &vars_samples, const IntResponseMap &resp_samples)
  computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank
- void archive_correlations (const StrStrSizet &run_identifier, ResultsManager &iterator_results, StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView dsv_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels) const
  save correlations to database
- bool correlations_computed () const
  returns corrComputed to indicate whether compute_correlations() has been invoked
- void print_correlations (std::ostream &s, StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView dsv_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels) const
  prints the correlations computed in compute_correlations()
### Private Member Functions

- **size_t find_valid_samples** (const IntResponseMap &resp_samples, BoolDeque &valid_sample)
  
  Find samples with finite response (any sample with any NaN or +/-Inf observation will be dropped)

- **void valid_sample_matrix** (const VariablesArray &vars_samples, const IntResponseMap &resp_samples, const StringSetArray &dss_vals, const BoolDeque is_valid_sample, RealMatrix &valid_data)
  
  Extract a compact valid sample (vars/resp) matrix from the passed data

- **void valid_sample_matrix** (const RealMatrix &vars_samples, const IntResponseMap &resp_samples, const BoolDeque is_valid_sample, RealMatrix &valid_samples)
  
  Extract a compact valid sample (vars/resp) matrix from the passed data

- **void values_to_ranks** (RealMatrix &valid_data)
  
  Replace sample values with their ranks, in-place

- **void center_rows** (RealMatrix &data_matrix)
  
  Center the passed matrix by its mean, in-place

- **void correl_adjust** (Real &corr_value)
  
  If result was NaN/Inf, preserve it, otherwise truncate to [-1.0, 1.0]

- **void simple_corr** (RealMatrix &total_data, const int &num_in, RealMatrix &corr_matrix)
  
  Computes simple correlations, populating corr_matrix

- **void partial_corr** (RealMatrix &total_data, const int num_in, const RealMatrix &simple_corr_mat, RealMatrix &corr_matrix, bool &numerical_issues)
  
  Computes partial correlations, populating corr_matrix and numerical_issues

- **bool has_nan_or_inf** (const RealMatrix &corr) const
  
  Return true if there are any NaN or Inf entries in the matrix.

### Static Private Member Functions

- **static bool rank_sort** (const int &x, const int &y)
  
  Sort algorithm to compute ranks for rank correlations

### Private Attributes

- **RealMatrix simpleCorr**
  
  Matrix to hold simple raw correlations

- **RealMatrix simpleRankCorr**
  
  Matrix to hold simple rank correlations

- **RealMatrix partialCorr**
  
  Matrix to hold partial raw correlations

- **RealMatrix partialRankCorr**
  
  Matrix to hold partial rank correlations

- **size_t numFns**
  
  Number of responses

- **size_t numVars**
  
  Number of inputs

- **bool numericalIssuesRaw**
  
  Flag indicating numerical issues in partial raw correlation calculations
• bool numericalIssuesRank
  flag indicating numerical issues in partial rank correlation calculations
• bool corrComputed
  flag indicating whether correlations have been computed

Static Private Attributes
• static RealArray rawData = RealArray()
  array to hold temporary data before sort

13.183.1 Detailed Description

Class for a utility class containing correlation calculations and variance-based decomposition.

This class provides code for several of the sampling methods both in the NonD branch and in the PStudyD-ACE branch. Currently, the utility functions provide global sensitivity analysis through correlation calculations (e.g. simple, partial, rank, raw) as well as variance-based decomposition.

13.183.2 Member Function Documentation

void compute_correlations ( const VariablesArray & vars_samples, const IntResponseMap & resp_samples, const StringSetArray & dss_vals )

computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank

This version is used when full variables objects are being processed. Calculates simple correlation, partial correlation, simple rank correlation, and partial rank correlation coefficients.

References Dakota::abort_handler(), SensAnalysisGlobal::corrComputed, SensAnalysisGlobal::find_valid_samples(), SensAnalysisGlobal::numericalIssuesRank, SensAnalysisGlobal::numericalIssuesRaw, SensAnalysisGlobal::numFns, SensAnalysisGlobal::numVars, SensAnalysisGlobal::partial_corr(), SensAnalysisGlobal::partialCorr, SensAnalysisGlobal::partialRankCorr, SensAnalysisGlobal::simple_corr(), SensAnalysisGlobal::simpleCorr, SensAnalysisGlobal::simpleRankCorr, SensAnalysisGlobal::valid_sample_matrix(), and SensAnalysisGlobal::values_to_ranks().

Referenced by NonDSampling::compute_statistics(), ParamStudy::post_run(), FSUDesignCompExp::post_run(), and DDACEDesignCompExp::post_run().

void compute_correlations ( const RealMatrix & vars_samples, const IntResponseMap & resp_samples )

computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank

This version is used when compact samples matrix is being processed. Calculates simple correlation, partial correlation, simple rank correlation, and partial rank correlation coefficients.

References Dakota::abort_handler(), SensAnalysisGlobal::corrComputed, SensAnalysisGlobal::find_valid_samples(), SensAnalysisGlobal::numericalIssuesRank, SensAnalysisGlobal::numericalIssuesRaw, SensAnalysisGlobal::numFns, SensAnalysisGlobal::numVars, SensAnalysisGlobal::partial_corr(), SensAnalysisGlobal::partialCorr, SensAnalysisGlobal::partialRankCorr, SensAnalysisGlobal::simple_corr(), SensAnalysisGlobal::simpleCorr, SensAnalysisGlobal::simpleRankCorr, SensAnalysisGlobal::valid_sample_matrix(), and SensAnalysisGlobal::values_to_ranks().

void values_to_ranks ( RealMatrix & valid_data ) [private]

replace sample values with their ranks, in-place

When converting values to ranks, uses the average ranks of any tied values

Referenced by SensAnalysisGlobal::compute_correlations().
void simple_corr ( RealMatrix & total_data, const int & num_in, RealMatrix & corr_matrix ) [private]

computes simple correlations, populating corr_matrix
Calculates simple correlation coefficients from a matrix of data (oriented factors x observations):

- num_corr is number of rows of total data
- num_in indicates whether only pairs of correlations should be calculated between pairs of columns (num_in vs. num_corr-num_in); if num_in = num_corr, correlations are calculated between all columns

References SensAnalysisGlobal::center_rows(), and SensAnalysisGlobal::correl_adjust().
Referenced by SensAnalysisGlobal::compute_correlations().

void partial_corr ( RealMatrix & total_data, const int num_in, const RealMatrix & simple_corr_mat, RealMatrix & corr_matrix, bool & numerical_issues ) [private]

computes partial correlations, populating corr_matrix and numerical_issues
Calculates partial correlation coefficients between num_in inputs and numRows() - num_in outputs.
References Dakota::abort_handler(), SensAnalysisGlobal::center_rows(), SensAnalysisGlobal::correl_adjust(), Dakota::qr(), Dakota::qr_rsolve(), and Dakota::svd().
Referenced by SensAnalysisGlobal::compute_correlations().
The documentation for this class was generated from the following files:

- SensAnalysisGlobal.hpp
- SensAnalysisGlobal.cpp

13.184 SeqHybridMetaIterator Class Reference

Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.

Inheritance diagram for SeqHybridMetaIterator:

```
Iterator

MetaIterator

SeqHybridMetaIterator
```

Public Member Functions

- **SeqHybridMetaIterator (ProblemDescDB &problem_db)**  
  *standard constructor*
- **SeqHybridMetaIterator (ProblemDescDB &problem_db, Model &model)**  
  *alternate constructor*
- **~SeqHybridMetaIterator ()**  
  *destructor*
CHAPTER 13. CLASS DOCUMENTATION

Protected Member Functions

- void core_run()
  Performs the hybrid iteration by executing a sequence of iterators, using a similar sequence of models that may vary in fidelity.
- void print_results(std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results
- void derived_init_communicators(ParLevLIter pliter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators(ParLevLIter pliter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators(ParLevLIter pliter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- IntIntPair estimate_partition_bounds()
  estimate the minimum and maximum partition sizes that can be utilized by this Iterator
- const Variables & variables_results() const
  return the final solution from selectedIterators(variables)
- const Response & response_results() const
  return the final solution from selectedIterators(response)
- void initialize_iterator(int job_index)
  used by IteratorScheduler to set the starting data for a run
- void pack_parameters_buffer(MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack starting data for an iterator run
- void unpack_parameters_initialize(MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack starting data and initialize an iterator run
- void pack_results_buffer(MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack results data from an iterator run
- void unpack_results_buffer(MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack results data from an iterator run
- void update_local_results(int job_index)
  used by IteratorScheduler to update local results arrays

Private Member Functions

- void run_sequential()
  run a sequential hybrid
- void run_sequential_adaptive()
  run a sequential adaptive hybrid
- void partition_sets(size_t num_sets, int job_index, size_t &start_index, size_t &job_size)
  convert num_sets and job_index into a start_index and job_size for extraction from parameterSets
- void extract_parameter_sets(int job_index, VariablesArray &partial_param_sets)
  extract partial_param_sets from parameterSets based on job_index
- void update_local_results(PPRArray &prp_results, int job_id)
  update the partial set of final results from the local iterator execution
- void initialize_iterator(const VariablesArray &param_sets)
  called by unpack_parameters_initialize(MPIUnpackBuffer) and initialize_iterator(int) to update the active Model and Iterator
Private Attributes

- String seqHybridType
  
  empty (default) or “adaptive”
- StringArray methodStrings
  
  the list of method pointer or method name identifiers
- StringArray modelStrings
  
  the list of model pointer identifiers for method identification by name
- bool lightwtMethodCtor
  
  use of lightweight Iterator construction by name
- bool singlePassedModel
  
  use of constructor that enforces use of a single passed Model
- IteratorArray selectedIterator
  
  the set of iterators, one for each entry in methodStrings
- ModelArray selectedModels
  
  the set of models, one for each iterator (if not lightweight construction)
- size_t seqCount
  
  hybrid sequence counter: 0 to numIterators-1
- Real progressThreshold
  
  when the progress metric falls below this threshold, the sequential adaptive hybrid switches to the next method
- PRP2DArray prpResults
  
  2-D array of results corresponding to numIteratorJobs, one set of results per job (iterators may return multiple final solutions)
- VariablesArray parameterSets
  
  1-D array of variable starting points for the iterator jobs

Friends

- class IteratorScheduler
  
  protect scheduler callback functions from general access

Additional Inherited Members

13.184.1 Detailed Description

Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.

Sequential hybrid meta-iteration supports two approaches: (1) the non-adaptive sequential hybrid runs one method to completion, passes its best results as the starting point for a subsequent method, and continues this succession until all methods have been executed (the stopping rules are controlled internally by each iterator), and (2) the adaptive sequential hybrid uses adaptive stopping rules for the iterators that are controlled externally by this method. Any iterator may be used so long as it defines the notion of a final solution which can be passed as starting data for subsequent iterators.
13.184.2 Member Function Documentation

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
Reimplemented from Iterator.
References Response::function_values(), Response::is_null(), Variables::is_null(), MetaIterator::iterSched, IteratorScheduler::messagePass, and SeqHybridMetaIterator::prpResults.

void run_sequential( ) [private]
run a sequential hybrid
In the sequential nonadaptive case, there is no interference with the iterators. Each runs until its own convergence criteria is satisfied. Status: fully operational.
References Iterator::accepts_multiple_points(), ParallelLibrary::bcast(), ParallelLibrary::bcast_hs(), Iterator::initialize_graphics(), Model::interface_id(), Iterator::iteratedModel, IteratorScheduler::iterator_message_lengths(), IteratorScheduler::iteratorCommRank, IteratorScheduler::iteratorCommSize, IteratorScheduler::iteratorScheduling, IteratorScheduler::iteratorServerId, MetaIterator::iterSched, IteratorScheduler::messagePass, Iterator::methodPCLiter, SeqHybridMetaIterator::methodStrings, IteratorScheduler::miPLIndex, Iterator::num_final_solutions(), IteratorScheduler::numIteratorJobs, IteratorScheduler::numIteratorServers, SeqHybridMetaIterator::pack_parameters_buffer(), Iterator::parallelLib, SeqHybridMetaIterator::parameterSets, SeqHybridMetaIterator::prpResults, ParallelLibrary::recv(), Iterator::response_results(), IteratorScheduler::schedule_iterators(), SeqHybridMetaIterator::selectedIterators, SeqHybridMetaIterator::selectedModels, SeqHybridMetaIterator::selectedIterators, Iterator::summaryOutputFlag, and Iterator::variables_results().
Referenced by SeqHybridMetaIterator::core_run().

void run_sequential_adaptive( ) [private]
run a sequential adaptive hybrid
In the sequential adaptive case, there is interference with the iterators through the use of the ++ overloaded operator. iterator++ runs the iterator for one cycle, after which a progress_metric is computed. This progress metric is used to dictate method switching instead of each iterator’s internal convergence criteria. Status: incomplete.
References Iterator::finalize_run(), Iterator::initialize_graphics(), Iterator::initialize_run(), IteratorScheduler::iteratorCommRank, IteratorScheduler::iteratorServerId, MetaIterator::iterSched, SeqHybridMetaIterator::methodStrings, IteratorScheduler::numIteratorServers, SeqHybridMetaIterator::progressThreshold, Iterator::response_results(), SeqHybridMetaIterator::selectedIterators, SeqHybridMetaIterator::selectedModels, SeqHybridMetaIterator::selectedIterators, SeqHybridMetaIterator::selectedModels, SeqHybridMetaIterator::seqCount, Iterator::summaryOutputFlag, and Iterator::variables_results().
Referenced by SeqHybridMetaIterator::core_run().

void extract_parameter_sets( int job_index, VariablesArray & partial_param_sets ) [inline], [private]
extract partial_param_sets from parameterSets based on job_index
This convenience function is executed on an iterator master (static scheduling) or a meta-iterator master (self scheduling) at run initialization time and has access to the full parameterSets array (this is All-Reduced for all peers at the completion of each cycle in run_sequential()).
References SeqHybridMetaIterator::parameterSets, and SeqHybridMetaIterator::partition_sets().
Referenced by SeqHybridMetaIterator::initialize_iterator(), and SeqHybridMetaIterator::pack_parameters_buffer(). The documentation for this class was generated from the following files:

- SeqHybridMetaIterator.hpp
- SeqHybridMetaIterator.cpp

13.185 SerialDirectApplicInterface Class Reference

Sample derived interface class for testing serial simulator plug-ins using assign_rep().

Inheritance diagram for SerialDirectApplicInterface:

```
+---------+
| Interface|
+---------+
    |          |
    | ApplicationInterface|
    |                     |
    +--------+          +---------+          +--------+
    |    DirectApplicInterface    | SerialDirectApplicInterface |
    +--------+    +---------+          +--------+
```

Public Member Functions

- SerialDirectApplicInterface (const Dakota::ProblemDescDB &problem_db)
  constructor
- ~SerialDirectApplicInterface ()
  destructor

Protected Member Functions

- int derived_map_ac (const Dakota::String &ac_name)
  execute an analysis code portion of a direct evaluation invocation
- void derived_map_async (const Dakota::ParamResponsePair &pair)
  no-op hides base error; job batching occurs within wait_local_evaluations()
- void wait_local_evaluations (Dakota::PRPQueue &prp_queue)
  evaluate the batch of jobs contained in prp_queue
- void test_local_evaluations (Dakota::PRPQueue &prp_queue)
  invokes wait_local_evaluations() (no special nowait support)
- void set_communicators_checks (int max_eval_concurrency)
  no-op hides default run-time error checks at DirectApplicInterface level

Private Member Functions

- int rosenbrock (const Dakota::RealVector &c_vars, short asv, Dakota::Real &fn_val, Dakota::RealVector &fn_grad, Dakota::RealSymMatrix &fn_hess)
  Rosenbrock plug-in test function.
Additional Inherited Members

13.185.1 Detailed Description

Sample derived interface class for testing serial simulator plug-ins using assign_rep(). The plug-in SerialDirectApplicInterface resides in namespace SIM and uses a copy of rosenbrock() to perform serial parameter to response mappings. It is used to demonstrate plugging in a serial direct analysis driver into Dakota in library mode. Test input files can then use an analysis driver of "plugin_rosenbrock".

13.185.2 Member Function Documentation

void test_local_evaluations ( Dakota::PRPQueue & prp_queue ) [inline], [protected]

invokes wait_local_evaluations() (no special nowait support)

For use by ApplicationInterface::serve_evaluations_asynch(), which can provide a batch processing capability within message passing schedulers (called using chain IteratorScheduler::run_iterator() -> Model::serve() -> ApplicationInterface::serve_evaluations() -> ApplicationInterface::serve_evaluations_asynch()). References SerialDirectApplicInterface::wait_local_evaluations().

The documentation for this class was generated from the following files:

- PluginSerialDirectApplicInterface.hpp
- PluginSerialDirectApplicInterface.cpp

13.186 TrackerHTTP::Server Struct Reference

struct to hold tracker/proxy pairs

Public Member Functions

- Server (std::string t, std::string p)

Public Attributes

- std::string tracker
- std::string proxy

13.186.1 Detailed Description

struct to hold tracker/proxy pairs

The documentation for this struct was generated from the following file:

- TrackerHTTP.hpp

13.187 SharedApproxData Class Reference

Base class for the shared approximation data class hierarchy.

Inheritance diagram for SharedApproxData:
Public Member Functions

- **SharedApproxData ()**
  default constructor
- **SharedApproxData (ProblemDescDB &problem_db, size_t num_vars)**
  standard constructor for envelope
- **SharedApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  alternate constructor for envelope
- **SharedApproxData (const SharedApproxData &approx)**
  copy constructor
- **virtual ~SharedApproxData ()**
  destructor
- **SharedApproxData operator= (const SharedApproxData &approx)**
  assignment operator
- **virtual void build (size_t index= NPOS)**
  builds the shared approximation data from scratch
- **virtual void rebuild (size_t index= NPOS)**
  rebuilds the shared approximation data incrementally
- **virtual void pop (bool save_surr_data)**
  back out the previous increment to the shared approximation data
- **virtual bool push_available ()**
  queries availability of pushing data associated with a trial set
- **virtual size_t retrieval_index ()**
  return index of trial set within popped bookkeeping sets
- **virtual void pre_push ()**
  push a previous state of the shared approximation data
- **virtual void post_push ()**
  clean up popped bookkeeping following push
- **virtual size_t finalization_index (size_t i)**
  return index of i-th trailing trial set within restorable bookkeeping sets
- **virtual void pre_finalize ()**
  finalize the shared approximation data following a set of increments
- **virtual void post_finalize ()**
  clean up popped bookkeeping following aggregation
- **virtual void store (size_t index= NPOS)**
  store the current state of the shared approximation data for later combination (defaults to push_back)
• virtual void restore (size_t index= NPOS)
  restore a previous state of the shared approximation data (defaults to pop_back from stored)
• virtual void remove_stored (size_t index= NPOS)
  remove an instance of stored approximation data prior to combination (defaults to pop_back)
• virtual size_t pre_combine ()
  aggregate the shared approximation data from current and stored states
• virtual void post_combine ()
  clean up stored data sets after aggregation
• virtual void clear_stored ()
  clear stored approximation data
• void set_bounds (const RealVector &c_l_bnds, const RealVector &c_u_bnds, const IntVector &di_l_bnds,
  const IntVector &di_u_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds)
  set approximation lower and upper bounds (currently only used by graphics)
• SharedApproxData * data_rep () const
  returns dataRep for access to derived class member functions that are not mapped to the top SharedApproxData level

Protected Member Functions

• SharedApproxData (BaseConstructor, ProblemDescDB &problem_db, size_t num_vars)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)
• SharedApproxData (NoDBBaseConstructor, const String &approx_type, size_t num_vars, short data_order,
  short output_level)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)

Protected Attributes

• size_t numVars
  number of variables in the approximation
• String approxType
  approximation type identifier
• short buildDataOrder
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format.
• short outputLevel
  output verbosity level: {SILENT,QUIET,NORMAL,VERBOSE,DEBUG}.OUTPUT
• String modelExportPrefix
  Prefix for model export files.
• unsigned short modelExportFormat
  Bitmapped format requests for exported models.
• RealVector approxCLowerBnds
  approximation continuous lower bounds (used by 3D graphics and Surfpack KrigingModel)
• RealVector approxCUpperBnds
  approximation continuous upper bounds (used by 3D graphics and Surfpack KrigingModel)
• IntVector approxDILowerBnds
  
  approximation continuous lower bounds

• IntVector approxDIUpperBnds
  
  approximation continuous upper bounds

• RealVector approxDRLowerBnds
  
  approximation continuous lower bounds

• RealVector approxDRUpperBnds
  
  approximation continuous upper bounds

Private Member Functions

• SharedApproxData * get_shared_data (ProblemDescDB &problem_db, size_t num_vars)
  
  Used only by the standard envelope constructor to initialize dataRep to the appropriate derived type.

• SharedApproxData * get_shared_data (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)
  
  Used only by the alternate envelope constructor to initialize dataRep to the appropriate derived type.

Private Attributes

• SharedApproxData * dataRep
  
  pointer to the letter (initialized only for the envelope)

• int referenceCount
  
  number of objects sharing dataRep

Friends

• class Approximation
• class TaylorApproximation
• class TANA3Approximation
• class GaussProcApproximation
• class VPSApproximation
• class SurfpackApproximation
• class PecosApproximation

13.187.1 Detailed Description

Base class for the shared approximation data class hierarchy.

The SharedApproxData class is the base class for the shared approximation data class hierarchy in DAKOTA. For memory efficiency and enhanced polymorphism, the approximation hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (SharedApproxData) serves as the envelope and one of the derived classes (selected in SharedApproxData::get_shared_data()) serves as the letter.
13.187.2 Constructor & Destructor Documentation

SharedApproxData( )

default constructor
   For the default constructor, dataRep is NULL. This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.
   Referenced by SharedApproxData::get_shared_data().

SharedApproxData ( ProblemDescDB & problem_db, size_t num_vars )

standard constructor for envelope
   Envelope constructor only needs to extract enough data to properly execute get_shared_data, since SharedApproxData(BaseConstructor, problem_db) builds the actual base class data for the derived approximations.
   References Dakota::abort_handler(), SharedApproxData::dataRep, and SharedApproxData::get_shared_data().

SharedApproxData ( const String & approx_type, const UShortArray & approx_order, size_t num_vars, short data_order, short output_level )

alternate constructor for envelope
   This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to problem_db, it utilizes the NoDBBaseConstructor constructor chain.
   References Dakota::abort_handler(), SharedApproxData::dataRep, and SharedApproxData::get_shared_data().

SharedApproxData ( const SharedApproxData & shared_data )

copy constructor
   Copy constructor manages sharing of dataRep and incrementing of referenceCount.
   References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

~SharedApproxData( ) [virtual]

destructor
   Destructor decrements referenceCount and only deletes dataRep when referenceCount reaches zero.
   References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

SharedApproxData ( BaseConstructor, ProblemDescDB & problem_db, size_t num_vars ) [protected]

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
   This constructor is the one which must build the base class data for all derived classes. get_shared_data() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_shared_data() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~SharedApproxData).
   References SharedApproxData::approxType, SharedApproxData::buildDataOrder, ProblemDescDB::get_bool(), ProblemDescDB::get_db_model_node(), ProblemDescDB::get_string(), ProblemDescDB::set_db_model_nodes(), Dakota::strbegins(), and Dakota::strends().
13.187. **SHAREDAPPROXDATA CLASS REFERENCE**

**SharedApproxData** (NoDBBaseConstructor, const String & approx_type, size_t num_vars, short data_order, short output_level) [protected]

Constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_shared_data() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_shared_data() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~SharedApproxData).

References SharedApproxData::approxType, SharedApproxData::buildDataOrder, Dakota::strbegins(), and Dakota::strends().

### 13.187.3 Member Function Documentation

**SharedApproxData operator= (const SharedApproxData & shared_data)**

Assignment operator

- References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

**SharedApproxData * get_shared_data (ProblemDescDB & problem_db, size_t num_vars)** [private]

Used only by the standard envelope constructor to initialize dataRep to the appropriate derived type.

- Used only by the envelope constructor to initialize dataRep to the appropriate derived type.
- References ProblemDescDB::get_string(), SharedApproxData::SharedApproxData(), and Dakota::strends().
- Referenced by SharedApproxData::SharedApproxData().

**SharedApproxData * get_shared_data (const String & approx_type, const UShortArray & approx_order, size_t num_vars, short data_order, short output_level)** [private]

Used only by the alternate envelope constructor to initialize dataRep to the appropriate derived type.

- Used only by the envelope constructor to initialize dataRep to the appropriate derived type.
- References SharedApproxData::SharedApproxData(), and Dakota::strends().

### 13.187.4 Member Data Documentation

**short buildDataOrder** [protected]

Order of the data used for surrogate construction, in ActiveSet request vector 3-bit format.

- This setting distinguishes derivative data intended for use in construction (includes derivatives w.r.t. the build variables) from derivative data that may be approximated separately (excludes derivatives w.r.t. auxiliary variables). This setting should also not be inferred directly from the responses specification, since we may need gradient support for evaluating gradients at a single point (e.g., the center of a trust region), but not require gradient evaluations at every point.
- Referenced by SharedSurfpackApproxData::add_sd_to_surfdata(), TaylorApproximation::build(), TaylorApproximation::gradient(), TaylorApproximation::hessian(), TaylorApproximation::min_coefficients(), Approximation::min_points(), Approximation::recommended_points(), SharedApproxData::SharedApproxData(), SharedPecosApproxData::SharedPecosApproxData(), SurfpackApproximation::SurfpackApproximation(), SurfpackApproximation::surrogates_to_surf_data(), TANA3Approximation::TANA3Approximation(), and TaylorApproximation::value().

The documentation for this class was generated from the following files:
• SharedApproxData.hpp
• SharedApproxData.cpp

13.188 SharedPecosApproxData Class Reference

Derived approximation class for global basis polynomials.
Inheritance diagram for SharedPecosApproxData:

```
SharedApproxData
    SharedPecosApproxData
```

Public Member Functions

- **SharedPecosApproxData ()**
  default constructor
- **SharedPecosApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  alternate constructor
- **SharedPecosApproxData (ProblemDescDB &problem_db, size_t num_vars)**
  standard ProblemDescDB-driven constructor
- **~SharedPecosApproxData ()**
  destructor
- **void random_variables_key (const Pecos::BitArray &random_vars_key)**
  set pecosBasisApprox.randomVarsKey
- **void integration_iterator (const Iterator &iterator)**
  set pecosBasisApprox.driverRep
- **void construct_basis (const Pecos::ShortArray &u_types, const Pecos::AleatoryDistParams &adp)**
  invoke Pecos::SharedOrthogPolyApproxData::construct_basis()
- **void polynomial_basis (const std::vector<Pecos::BasisPolynomial> &poly_basis)**
  set Pecos::SharedOrthogPolyApproxData::polynomialBasis
- **const std::vector<Pecos::BasisPolynomial> & polynomial_basis () const**
  get Pecos::SharedOrthogPolyApproxData::polynomialBasis
- **std::vector < Pecos::BasisPolynomial > & polynomial_basis ()**
  get Pecos::SharedOrthogPolyApproxData::polynomialBasis
- **void allocate (const UShort2DArray &mi)**
  set Pecos::SharedOrthogPolyApproxData::multiIndex and allocate associated arrays
- **const UShort2DArray & multi_index () const**
  get Pecos::SharedOrthogPolyApproxData::multiIndex
- **const Pecos::BitArrayULongMap & sobol_index_map () const**
return Pecos::SharedPolyApproxData::sobolIndexMap

- void coefficients_norms_flag (bool flag)
  
  invokes Pecos::SharedOrthogPolyApproxData::coefficients_norms_flag()

- size_t expansion_terms () const
  
  return Pecos::SharedOrthogPolyApproxData::expansion_terms()

- const UShortArray & expansion_order () const
  
  return Pecos::SharedOrthogPolyApproxData::expansion_order()

- void expansion_order (const UShortArray &order)
  
  invokes Pecos::SharedOrthogPolyApproxData::expansion_order(UShortArray&)

- void increment_order ()
  
  invokes Pecos::SharedOrthogPolyApproxData::increment_order()

- void configuration_options (const Pecos::ExpansionConfigOptions &ec_options)
  
  set the expansion configuration options within Pecos::SharedPolyApproxData

- void configuration_options (const Pecos::BasisConfigOptions &bc_options)
  
  set the basis configuration options within Pecos::SharedPolyApproxData

- void configuration_options (const Pecos::RegressionConfigOptions &rc_options)
  
  set the regression configuration options within Pecos::SharedRegressOrthogPolyApproxData

Protected Member Functions

- void build (size_t index=_NPOS)
  
  builds the shared approximation data from scratch

- void rebuild (size_t index=_NPOS)
  
  rebuilds the shared approximation data incrementally

- void pop (bool save_surr_data)
  
  back out the previous increment to the shared approximation data

- bool push_available ()
  
  queries availability of pushing data associated with a trial set

- size_t retrieval_index ()
  
  return index of trial set within popped bookkeeping sets

- void pre_push ()
  
  push a previous state of the shared approximation data

- void post_push ()
  
  clean up popped bookkeeping following push

- size_t finalization_index (size_t i)
  
  return index of i-th trailing trial set within restorable bookkeeping sets

- void pre_finalize ()
  
  finalize the shared approximation data following a set of increments

- void post_finalize ()
  
  clean up popped bookkeeping following aggregation

- void store (size_t index=_NPOS)
  
  store the current state of the shared approximation data for later combination (defaults to push_back)

- void restore (size_t index=_NPOS)
Private Member Functions

- Pecos::SharedBasisApproxData & pecos_shared_data ()
  - return pecosSharedData
- void approx_type_to_basis_type (const String &approx_type, short &basis_type)
  - utility to convert Dakota type string to Pecos type enumeration

Private Attributes

- Pecos::SharedBasisApproxData pecosSharedData
  - the Pecos shared approximation data
- Pecos::SharedPolyApproxData * pecosSharedDataRep
  - convenience pointer to derived letter within pecosSharedData

Friends

- class PecosApproximation

Additional Inherited Members

13.188.1 Detailed Description

Derived approximation class for global basis polynomials.

The SharedPecosApproxData class provides a global approximation based on basis polynomials. This includes orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic collocation.

The documentation for this class was generated from the following files:

- SharedPecosApproxData.hpp
- SharedPecosApproxData.cpp

13.189 SharedResponseData Class Reference

Container class encapsulating variables data that can be shared among a set of Response instances.
Public Member Functions

- **SharedResponseData ()**
  default constructor

- **SharedResponseData (const ProblemDescDB &problem_db)**
  standard constructor

- **SharedResponseData (const ActiveSet &set)**
  alternate on-the-fly constructor (explicit disallows its use for implicit type conversion so that instantiations of Response(set) are invalid)

- **SharedResponseData (const SharedResponseData &srd)**
  copy constructor

- **~SharedResponseData ()**
  destructor

- **SharedResponseData & operator= (const SharedResponseData &srd)**
  assignment operator

- **bool operator== (const SharedResponseData &other)**
  experimental operator== for use in unit testing

- **size_t num_scalar_responses () const**
  number of scalar responses

- **size_t num_field_response_groups () const**
  number of field response groups

- **size_t num_response_groups () const**
  total number of response groups (scalars + field groups)

- **size_t num_field_functions () const**
  total number of field functions (1-norm of fieldRespGroupLengths)

- **size_t num_functions () const**
  total number of response functions (scalars + 1-norm of fieldRespGroupLengths)

- **const IntVector & field_lengths () const**
  index of field lengths for field data

- **void field_lengths (const IntVector &field_lengths)**
  set field lengths (if experiment different from simulation)

- **const String & function_label (size_t i) const**
  return a response function identifier string

- **const StringArray & function_labels () const**
  return the response function identifier strings

- **StringArray & function_labels ()**
  return the response function identifier strings

- **void function_label (const String &label, size_t i)**
  set a response function identifier string

- **void function_labels (const StringArray &labels)**
  set the response function identifier strings

- **const StringArray & field_group_labels ()**
return the coarse (per-group) field response labels

- void field_group_labels (const StringArray &field_labels)
  set the coarse field group labels (must agree with number fields)

- const String & responses_id () const
  return the response identifier

- short response_type () const
  return the response type: \{BASE,SIMULATION,EXPERIMENT\}_RESPONSE

- void response_type (short type)
  set the response type: \{BASE,SIMULATION,EXPERIMENT\}_RESPONSE

- short primary_fn_type () const
  get the primary function type (generic, objective, calibration)

- void primary_fn_type (short type)
  set the primary function type (generic, objective, calibration)

- const RealVector & simulation_error () const
  retrieve simulation variance

- SharedResponseData copy () const
  create a deep copy of the current object and return by value

- void reshape (size_t num_fns)
  reshape the data, disconnecting a shared rep if necessary

- bool is_null () const
  return true if empty handle with null representation

- long reference_count () const
  how many handles (including this) are sharing this representation (body); for debugging/testing only

- template<class Archive>
  void serialize (Archive &ar, const unsigned int version)
  serialize through the pointer, which requires object tracking: write and read are symmetric for this class

Private Member Functions

- template<class Archive>
  void serialize (Archive &ar, const unsigned int version)
  serialize through the pointer, which requires object tracking: write and read are symmetric for this class

Private Attributes

- boost::shared_ptr
  < SharedResponseDataRep > srdRep
  pointer to the body (handle-body idiom)

Friends

- class boost::serialization::access
  allow boost access to serialize this class
13.189.1 Detailed Description

Container class encapsulating variables data that can be shared among a set of Response instances.

An array of Response objects (e.g., Analyzer::allResponse) contains repeated configuration data (id’s, labels, counts). SharedResponseData employs a handle-body idiom to allow this shared data to be managed in a single object with many references to it, one per Response object in the array. This allows scaling to larger sample sets.

13.189.2 Member Function Documentation

SharedResponseData copy ( ) const

create a deep copy of the current object and return by value

Deep copies are used when recasting changes the nature of a Response set.

References SharedResponseData::srdRep.

Referenced by ExperimentData::initialize(), and ExperimentData::load_data().

The documentation for this class was generated from the following files:

- SharedResponseData.hpp
- SharedResponseData.cpp

13.190 SharedResponseDataRep Class Reference

The representation of a SharedResponseData instance. This representation, or body, may be shared by multiple SharedResponseData handle instances.

Public Member Functions

- ~SharedResponseDataRep ()
  destructor must be public for shared ptr
- template<class Archive>
  void serialize (Archive &ar, const unsigned int version)

Private Member Functions

- SharedResponseDataRep ()
  default constructor
- SharedResponseDataRep (const ProblemDescDB &problem_db)
  standard constructor
- SharedResponseDataRep (const ActiveSet &set)
  alternate on-the-fly constructor
- void copy_rep (SharedResponseDataRep *srd_rep)
  copy the data from srd_rep to the current representation
- template<class Archive>
  void serialize (Archive &ar, const unsigned int version)
  serialize the core shared response data: write and read are symmetric for this class
- bool operator== (const SharedResponseDataRep &other)
  experimental operator== for use in unit testing
- void build_field_labels ()
  build/update the unrolled field labels based on fieldLabels and group lengths
Private Attributes

- short responseType
  enumeration of BASE_RESPONSE, SIMULATION_RESPONSE, or EXPERIMENT_RESPONSE
- short primaryFnType
  data set type for primary response: generic, objective, calibration
- String responsesId
  response identifier string from the input file
- StringArray functionLabels
  fine-grained (unrolled) set of response function identifiers used to improve output readability; length Response-::functionValues
- StringArray fieldLabels
  labels for each field group
- RealVector simulationVariance
  simulation variance
- size_t numScalarResponses
  number of scalar responses
- IntVector fieldRespGroupLengths
  index of field lengths for field data
- IntVector numCoordsPerField
  dimensions of each function

Friends

- class SharedResponseData
- class boost::serialization::access
  allow boost access to serialize this class

13.190.1 Detailed Description

The representation of a SharedResponseData instance. This representation, or body, may be shared by multiple SharedResponseData handle instances.

The SharedResponseData/SharedResponseDataRep pairs utilize a handle-body idiom (Coplien, Advanced C++).

13.190.2 Member Function Documentation

void copy_rep (SharedResponseDataRep * srd_rep) [private]

copy the data from srd_rep to the current representation

Deep copies are used when recasting changes the nature of a Response set.

References SharedResponseDataRep::fieldLabels, SharedResponseDataRep::fieldRespGroupLengths, SharedResponseDataRep::functionLabels, SharedResponseDataRep::numCoordsPerField, SharedResponseDataRep::numScalarResponses, SharedResponseDataRep::primaryFnType, SharedResponseDataRep::responsesId, SharedResponseDataRep::responseType, and SharedResponseDataRep::simulationVariance.

The documentation for this class was generated from the following files:

- SharedResponseData.hpp
- SharedResponseData.cpp
13.191 SharedSurfpackApproxData Class Reference

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

Inheritance diagram for SharedSurfpackApproxData:

```
SharedSurfpackApproxData
    \downarrow
SharedApproxData
```

Public Member Functions

- `SharedSurfpackApproxData ()`
  *default constructor*
- `SharedSurfpackApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)`
  *alternate constructor*
- `SharedSurfpackApproxData (ProblemDescDB &problem_db, size_t num_vars)`
  *standard constructor: Surfpack surface of appropriate type will be created*
- `~SharedSurfpackApproxData ()`
  *destructor*

Private Member Functions

- `void add_sd_to_surfdata (const Pecos::SurrogateDataVars &sdv, const Pecos::SurrogateDataResp &sdr, short fail_code, SurfData &surf_data)`
  *add Pecos::SurrogateData::SurrogateData{Vars,Resp} to SurfData, accounting for buildDataOrder available*
- `void copy_matrix (const RealSymMatrix &rsm, SurfpackMatrix<Real> &surfpack_matrix)`
  *copy RealSymMatrix to SurfpackMatrix (Real type only)*
- `void merge_variable_arrays (const RealVector &cv, const IntVector &div, const RealVector &drv, RealArray &ra)`
  *merge cv, div, and drv vectors into a single ra array*
- `void sdv_to_realarray (const Pecos::SurrogateDataVars &sdv, RealArray &ra)`
  *aggregate {continuous,discrete int,discrete real} variables from SurrogateDataVars into ra*
- `void vars_to_realarray (const Variables &vars, RealArray &ra)`
  *aggregate {active,all} {continuous,discrete int,discrete real} variables into ra*

Private Attributes

- `unsigned short approxOrder`
  *order of polynomial approximation*
- `StringArray diagnosticSet`
  *set of diagnostic metrics*
- `bool crossValidateFlag`
whether to perform cross validation

- unsigned numFolds
  number of folds for CV
- Real percentFold
  percentage of data for CV
- bool pressFlag
  whether to perform PRESS

Friends

- class SurfpackApproximation
- class VPSApproximation

Additional Inherited Members

13.191.1 Detailed Description

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

The SharedSurfpackApproxData class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SharedSurfpackApproxData builds a Surfpack Surface object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial neural networks, radial basis function network, or multivariate adaptive regression splines (MARS).

13.191.2 Constructor & Destructor Documentation

SharedSurfpackApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)

alternate constructor

On-the-fly constructor which uses mostly Surfpack model defaults.

References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, and SharedApproxData::approxType.

SharedSurfpackApproxData (ProblemDescDB &problem_db, size_t num_vars)

standard constructor: Surfpack surface of appropriate type will be created

Initialize the embedded Surfpack surface object and configure it using the specifications from the input file. Data for the surface is created later.

References SharedSurfpackApproxData::approxOrder, SharedApproxData::approxType, ProblemDescDB::get_short(), and ProblemDescDB::get_string().

The documentation for this class was generated from the following files:

- SharedSurfpackApproxData.hpp
- SharedSurfpackApproxData.cpp

13.192 SharedVariablesData Class Reference

Container class encapsulating variables data that can be shared among a set of Variables instances.
Public Member Functions

- `SharedVariablesData()`  
  default constructor

- `SharedVariablesData(const ProblemDescDB &problem_db, const std::pair<short, short> &view)`  
  standard constructor

- `SharedVariablesData(const std::pair<short, short> &view, const std::map<unsigned short, size_t>&vars_comps, const BitArray&all_relax_dr=BitArray(), const BitArray&all_relax_di=BitArray())`  
  medium weight constructor providing detailed variable counts

- `SharedVariablesData(const std::pair<short, short> &view, const SizetArray&vars_comps_totals, const BitArray&all_relax_dr=BitArray(), const BitArray&all_relax_di=BitArray())`  
  lightweight constructor providing variable count totals

- `SharedVariablesData(const SharedVariablesData&svd)`  
  copy constructor

- `~SharedVariablesData()`  
  destructor

- `SharedVariablesData&operator=(const SharedVariablesData&svd)`  
  assignment operator

- `SharedVariablesData copy()const`  
  create a deep copy of the current object and return by value

- `void all_counts(size_t&num_acv, size_t&num_adiv, size_t&num_adsv, size_t&num_adrv) const`  
  compute all variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int,Real}

- `void design_counts(size_t&num_cdv, size_t&num_ddiv, size_t&num_ddsv, size_t&num_ddrv) const`  
  compute design variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int,Real}

- `void aleatory_uncertain_counts(size_t&num_cauv, size_t&num_dauiv, size_t&num_dausv, size_t&num_dauurv) const`  
  compute aleatory uncertain variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int,Real}

- `void epistemic_uncertain_counts(size_t&num_ceuv, size_t&num_deuiv, size_t&num_deusv, size_t&num_deurv) const`  
  compute epistemic uncertain variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int,Real}

- `void uncertain_counts(size_t&num_cuv, size_t&num_duiv, size_t&num_dusv, size_t&num_durv) const`  
  compute uncertain variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int,Real}

- `void state_counts(size_t&num_csv, size_t&num_dsv, size_t&num_dsvv, size_t&num_dsrv) const`  
  compute state variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int,Real}

- `void initialize_active_start_counts()`  
  initialize start index and counts for active variables

- `void initialize_inactive_start_counts()`  
  initialize start index and counts for inactive variables

- `void initialize_active_components()`
CHAPTER 13. CLASS DOCUMENTATION

- **void initialize_inactive_components()**
  - initialize the inactive components totals given inactive variable counts
- **const BitArray & all_relaxed_discrete_int() const**
  - return allRelaxedDiscreteInt
- **const BitArray & all_relaxed_discrete_real() const**
  - return allRelaxedDiscreteReal
- **StringMultiArrayView all_continuous_labels(size_t start, size_t num_items) const**
  - get num_items continuous labels beginning at index start
- **void all_continuous_labels(StringMultiArrayConstView cv_labels, size_t start, size_t num_items)**
  - set num_items continuous labels beginning at index start
- **void all_continuous_label(const String &cv_label, size_t index)**
  - set continuous label at index start
- **StringMultiArrayView all_discrete_int_labels(size_t start, size_t num_items) const**
  - get num_items discrete integer labels beginning at index start
- **void all_discrete_int_labels(StringMultiArrayConstView div_labels, size_t start, size_t num_items)**
  - set num_items discrete integer labels beginning at index start
- **void all_discrete_int_label(const String &div_label, size_t index)**
  - set discrete integer label at index start
- **StringMultiArrayView all_discrete_string_labels(size_t start, size_t num_items) const**
  - get num_items discrete string labels beginning at index start
- **void all_discrete_string_labels(StringMultiArrayConstView dsv_labels, size_t start, size_t num_items)**
  - set num_items discrete string labels beginning at index start
- **void all_discrete_string_label(const String &dsv_label, size_t index)**
  - set discrete string label at index start
- **UShortMultiArrayConstView all_continuous_types(size_t start, size_t num_items) const**
  - get num_items continuous types beginning at index start
- **void all_continuous_types(UShortMultiArrayConstView cv_types, size_t start, size_t num_items)**
  - set num_items continuous types beginning at index start
- **void all_continuous_type(unsigned short cv_type, size_t index)**
  - set continuous type at index
- **UShortMultiArrayConstView all_discrete_int_types(size_t start, size_t num_items) const**
  - get num_items discrete integer types beginning at index start
- **void all_discrete_int_types(UShortMultiArrayConstView div_types, size_t start, size_t num_items)**
  - set num_items discrete integer types beginning at index start
- **void all_discrete_int_type(unsigned short div_type, size_t index)**
  - set discrete integer type at index
• UShortMultiArrayConstView all_discrete_string_types (size_t start, size_t num_items) const
  get num_items discrete string types beginning at index start
• void all_discrete_string_types (UShortMultiArrayConstView dsv_types, size_t start, size_t num_items)
  set num_items discrete string types beginning at index start
• void all_discrete_string_type (unsigned short dsv_type, size_t index)
  set discrete string type at index
• UShortMultiArrayConstView all_discrete_real_types (size_t start, size_t num_items) const
  get num_items discrete real types beginning at index start
• void all_discrete_real_types (UShortMultiArrayConstView drv_types, size_t start, size_t num_items)
  set num_items discrete real types beginning at index start
• void all_discrete_real_type (unsigned short drv_type, size_t index)
  set discrete real type at index
• SizetMultiArrayConstView all_continuous_ids (size_t start, size_t num_items) const
  get num_items continuous ids beginning at index start
• void all_continuous_ids (SizetMultiArrayConstView cv_ids, size_t start, size_t num_items)
  set num_items continuous ids beginning at index start
• void all_continuous_id (size_t id, size_t index)
  set num_items continuous ids beginning at index start
• const String & id () const
  return the user-provided or default Variables identifier
• const SizetArray & components_totals () const
  return variable type counts for {continuous, discrete integer, discrete real} {design, aleatory uncertain, epistemic uncertain, state}
• const SizetArray & active_components_totals () const
  return active variable type counts for {continuous, discrete integer, discrete real} {design, aleatory uncertain, epistemic uncertain, state}
• const SizetArray & inactive_components_totals () const
  return inactive variable type counts for {continuous, discrete integer, discrete real} {design, aleatory uncertain, epistemic uncertain, state}
• size_t vc_lookup (unsigned short key) const
  retrieve the variables type count within svdRep->variablesComponents corresponding to (a fine-grain variables type) key
• const std::pair< short, short > & view () const
  retrieve the Variables view
• void inactive_view (short view2)
  set the inactive Variables view
• size_t cv () const
  get number of active continuous vars
• size_t cv_start () const
  get start index of active continuous vars
• size_t div () const
  get number of active discrete int vars
• size_t div_start () const
  get start index of active discrete int vars
• `size_t dsv () const`
  get number of active discrete string vars
• `size_t dsv_start () const`
  get start index of active discrete string vars
• `size_t drv () const`
  get number of active discrete real vars
• `size_t drv_start () const`
  get start index of active discrete real vars
• `size_t icv () const`
  get number of inactive continuous vars
• `size_t icv_start () const`
  get start index of inactive continuous vars
• `size_t idiv () const`
  get number of inactive discrete int vars
• `size_t idiv_start () const`
  get start index of inactive discrete int vars
• `size_t idsv () const`
  get number of inactive discrete string vars
• `size_t idsv_start () const`
  get start index of inactive discrete string vars
• `size_t idrv () const`
  get number of inactive discrete real vars
• `size_t idrv_start () const`
  get start index of inactive discrete real vars
• `void cv (size_t ncv)`
  set number of active continuous vars
• `void cv_start (size_t cvs)`
  set start index of active continuous vars
• `void div (size_t ndiv)`
  set number of active discrete int vars
• `void div_start (size_t divs)`
  set start index of active discrete int vars
• `void dsv (size_t ndsv)`
  set number of active discrete string vars
• `void dsv_start (size_t dsvs)`
  set start index of active discrete string vars
• `void drv (size_t ndrv)`
  set number of active discrete real vars
• `void drv_start (size_t drvs)`
  set start index of active discrete real vars
• `void icv (size_t nicv)`
  set number of inactive continuous vars
• `void icv_start (size_t icvs)`
set start index of inactive continuous vars

- void idiv (size_t nidiv)
  set number of inactive discrete int vars
- void idiv_start (size_t idivs)
  set start index of inactive discr int vars
- void idsv (size_t nidsv)
  set number of inactive discr string vars
- void idsv_start (size_t idsvs)
  set start index of inact discr string vars
- void idrv (size_t nidrv)
  set number of inactive discrete real vars
- void idrv_start (size_t idrvs)
  set start index of inact discr real vars

- template <class Archive>
  void serialize (Archive &ar, const unsigned int version)

**Private Member Functions**

- template <class Archive>
  void serialize (Archive &ar, const unsigned int version)
  serialize through the pointer, which requires object tracking: write and read are symmetric for this class

**Private Attributes**

- boost::shared_ptr
  < SharedVariablesDataRep > svdRep
  pointer to the body (handle-body idiom)

**Friends**

- class boost::serialization::access
  allow boost access to serialize this class

### 13.192.1 Detailed Description

Container class encapsulating variables data that can be shared among a set of Variables instances.

An array of Variables objects (e.g., Analyzer::allVariables) contains repeated configuration data (id’s, labels, counts). SharedVariablesData employs a handle-body idiom to allow this shared data to be managed in a single object with many references to it, one per Variables object in the array. This allows scaling to larger sample sets.

### 13.192.2 Member Function Documentation

**SharedVariablesData copy ( ) const**
create a deep copy of the current object and return by value

Deep copies are used when recasting changes the nature of a Variables set. References Dakota::svd(), and SharedVariablesData::svdRep.

The documentation for this class was generated from the following files:
13.193  **SharedVariablesDataRep** Class Reference

The representation of a **SharedVariablesData** instance. This representation, or body, may be shared by multiple **SharedVariablesData** handle instances.

**Public Member Functions**

- `~SharedVariablesDataRep()`
  - Destructor must be public for shared ptr
- `template<class Archive>
  void save(Archive &ar, const unsigned int version) const`
- `template<class Archive>
  void load(Archive &ar, const unsigned int version)`

**Private Member Functions**

- `SharedVariablesDataRep(const ProblemDescDB &problem, const std::pair<short, short> &view)`
  - Standard constructor
- `SharedVariablesDataRep(const std::pair<short, short> &view, const std::map<unsigned short, size_t> &vars_comps, const BitArray &all_relax_di, const BitArray &all_relax_dr)`
  - Medium weight constructor providing detailed variable counts
- `SharedVariablesDataRep(const std::pair<short, short> &view, const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr)`
  - Lightweight constructor providing variable count totals
- `SharedVariablesDataRep()`
  - Default constructor
- `void initialize_components_totals(const ProblemDescDB &problem_db)`
  - Populate variables\{Components,CompsTotals\} from user variable type and count specifications
- `void components_to_totals()`
  - Update variables\{Components,CompsTotals\} from variablesComponents
- `void relax_noncategorical(const ProblemDescDB &problem_db)`
  - Populate all\{RelaxedDiscrete\{Int,Real\} from user specifications (relax variables that are not declared as categorical)
- `void set_relax(const BitArray &user_cat_spec, size_t ucs_index, size_t ard_cntntr, BitArray &ard_container)`
  - Set the ard_cntntr entry in the all-relaxed-discrete integer or real container ard_container, based on user-specification of categorical, accounting for empty.
- `void all_counts(size_t &num_acv, size_t &num_adiv, size_t &num_ads, size_t &num_adrv) const`
  - Compute all variables sums from variablesCompsTotals
- `void relax_counts(size_t &num_cv, size_t &num_div, size_t &num_ds, size_t &num_dr, size_t &offset_di, size_t &offset_dr)`
  - Adjust counts based on all\{RelaxedDiscrete\{Int,Real\}
- `void design_counts(size_t &num_cdv, size_t &num_ddiv, size_t &num_ddv, size_t &num_ddrv)`

---

- `SharedVariablesData.hpp`
- `SharedVariablesData.cpp`
compute design variables sums from variablesCompsTotals

- **void aleatory_uncertain_counts** (size_t &num_cauv, size_t &num_dauiv, size_t &num_dausv, size_t &num_daurv) const
  compute aleatory uncertain variables sums from variablesCompsTotals

- **void epistemic_uncertain_counts** (size_t &num_ceuv, size_t &num_deuiv, size_t &num_deusv, size_t &num_deurv) const
  compute epistemic uncertain variables sums from variablesCompsTotals

- **void uncertain_counts** (size_t &num_cuv, size_t &num_duv, size_t &num_dusv, size_t &num_durv) const
  compute uncertain variables sums from variablesCompsTotals

- **void state_counts** (size_t &num_csv, size_t &num_dsv, size_t &num_dssv, size_t &num_dsrv) const
  compute state variables sums from variablesCompsTotals

- **void view_start_counts** (short view, size_t &cv_start, size_t &div_start, size_t &dsv_start, size_t &drv_start, size_t &num_cv, size_t &num_div, size_t &num_dsv, size_t &num_drv) const
  define start indices and counts for active variables based on view

- **void size_all_labels** ()
  size all{Continuous,DiscreteInt,DiscreteString,DiscreteReal}Labels, with or without discrete relaxation

- **void size_all_types** ()
  size all{Continuous,DiscreteInt,DiscreteString,DiscreteReal}Types, with or without discrete relaxation

- **void initialize_all_labels** (const ProblemDescDB &problem_db)
  aggregate all{Continuous,DiscreteInt,DiscreteString,DiscreteReal}Labels from user specification or defaults

- **void initialize_all_types** ()
  initialize all{Continuous,DiscreteInt,DiscreteString,DiscreteReal}Types, with or without discrete relaxation

- **void initialize_all_ids** ()
  initialize allContinuousIds (discrete not currently needed), with or without discrete relaxation

- **void initialize_active_start_counts** ()
  initialize {c,di,dr}vStart and num{D,DI,DS,DR}V

- **void initialize_inactive_start_counts** ()
  initialize {c,di,dr}vStart and num{D,DI,DS,DR}V

- **void initialize_active_components** ()
  initialize activeVarsCompsTotals given {c,di,dr}vStart and num{C,DI,DR}V

- **void initialize_inactive_components** ()
  initialize inactiveVarsCompsTotals given i{c,di,dr}vStart and numI{C,DI,DR}V

- **size_t vc_lookup** (unsigned short key) const
  retrieve the count within variablesComponents corresponding to key

- **void copy_rep** (SharedVariablesDataRep *svd_rep)
  copy the data from svd_rep to the current representation

- **template<class Archive>**
  void save (Archive &ar, const unsigned int version) const
  serialize the core shared variables data

- **template<class Archive>**
  void load (Archive &ar, const unsigned int version)
  load the core shared variables data and restore class state

- **BOOST_SERIALIZATION_SPLIT_MEMBER** () String variablesId
  variables identifier string from the input file
Private Attributes

- std::map< unsigned short, size_t > variablesComponents
  map linking variable types to counts
- SizetArray variablesCompsTotals
  totals for variable type counts for { continuous, discrete integer, discrete string, discrete real } { design, aleatory uncertain, epistemic uncertain, state }.
- SizetArray activeVarsCompsTotals
  totals for active variable type counts for { continuous, discrete integer, discrete string, discrete real } { design, aleatory uncertain, epistemic uncertain, state }.
- SizetArray inactiveVarsCompsTotals
  totals for inactive variable type counts for { continuous, discrete integer, discrete string, discrete real } { design, aleatory uncertain, epistemic uncertain, state }.
- std::pair< short, short > variablesView
  the variables view pair containing active (first) and inactive (second) view enumerations
- size_t cvStart
  start index of active continuous variables within allContinuousVars
- size_t divStart
  start index of active discrete integer variables within allDiscreteIntVars
- size_t dsvStart
  start index of active discrete string vars within allDiscreteStringVars
- size_t drvStart
  start index of active discrete real variables within allDiscreteRealVars
- size_t icvStart
  start index of inactive continuous variables within allContinuousVars
- size_t idivStart
  start index of inactive discrete integer vars within allDiscreteIntVars
- size_t idsvStart
  start index of inactive discrete string vars within allDiscreteStringVars
- size_t idrvStart
  start index of inactive discrete real variables within allDiscreteRealVars
- size_t numCV
  number of active continuous variables
- size_t numDIV
  number of active discrete integer variables
- size_t numDSV
  number of active discrete string variables
- size_t numDRV
  number of active discrete real variables
- size_t numICV
  number of inactive continuous variables
- size_t numIDIV
  number of inactive discrete integer variables
- size_t numIDSV
number of inactive discrete string variables
• size_t numIDRV

number of inactive discrete real variables
• StringMultiArray allContinuousLabels
  array of variable labels for all of the continuous variables
• StringMultiArray allDiscreteIntLabels
  array of variable labels for all of the discrete integer variables
• StringMultiArray allDiscreteStringLabels
  array of variable labels for all of the discrete string variables
• StringMultiArray allDiscreteRealLabels
  array of variable labels for all of the discrete real variables
• UShortMultiArray allContinuousTypes
  array of variable types for all of the continuous variables
• UShortMultiArray allDiscreteIntTypes
  array of variable types for all of the discrete integer variables
• UShortMultiArray allDiscreteStringTypes
  array of variable types for all of the discrete string variables
• UShortMultiArray allDiscreteRealTypes
  array of variable types for all of the discrete real variables
• SizetMultiArray allContinuousIds
  array of 1-based position identifiers for the all continuous variables array
• BitArray allRelaxedDiscreteInt
  array of booleans to indicate relaxation (promotion from DiscreteInt to Continuous) for all specified discrete int
  variables Note: container will be empty when not relaxing variables
• BitArray allRelaxedDiscreteReal
  array of booleans to indicate relaxation (promotion from DiscreteReal to Continuous) for all specified discrete real
  variables Note: container will be empty when not relaxing variables

Friends

• class SharedVariablesData
• class boost::serialization::access
  allow boost access to serialize this class

13.193.1 Detailed Description

The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple
SharedVariablesData handle instances.

The SharedVariablesData/SharedVariablesDataRep pairs utilize a handle-body idiom (Coplien, Advanced
C++).
CHAPTER 13. CLASS DOCUMENTATION

13.193.2 Member Function Documentation

```c++
void copy_rep ( SharedVariablesDataRep * svd_rep ) [private]
```
copy the data from svd_rep to the current representation

Deep copies are used when recasting changes the nature of a Variables set.

References SharedVariablesDataRep::activeVarsCompsTotals, SharedVariablesDataRep::allContinuousIds, SharedVariablesDataRep::allContinuousLabels, SharedVariablesDataRep::allContinuousTypes, SharedVariablesDataRep::allDiscreteIntLabels, SharedVariablesDataRep::allDiscreteIntTypes, SharedVariablesDataRep::allDiscreteRealLabels, SharedVariablesDataRep::allDiscreteRealTypes, SharedVariablesDataRep::allDiscreteStringLabels, SharedVariablesDataRep::allDiscreteStringTypes, SharedVariablesDataRep::allDiscreteRealTypes, SharedVariablesDataRep::allRelaxedDiscreteInt, SharedVariablesDataRep::allRelaxedDiscreteReal, SharedVariablesDataRep::cvStart, SharedVariablesDataRep::divStart, SharedVariablesDataRep::drvStart, SharedVariablesDataRep::dsvStart, SharedVariablesDataRep::icvStart, SharedVariablesDataRep::idivStart, SharedVariablesDataRep::idrvStart, SharedVariablesDataRep::idsvStart, SharedVariablesDataRep::inactiveVarsCompsTotals, SharedVariablesDataRep::numCV, SharedVariablesDataRep::numDIV, SharedVariablesDataRep::numDRC, SharedVariablesDataRep::numDSV, SharedVariablesDataRep::numICV, SharedVariablesDataRep::numDIV, SharedVariablesDataRep::numDRC, SharedVariablesDataRep::numDSV, SharedVariablesDataRep::variablesComponents, SharedVariablesDataRep::variablesCompsTotals, and SharedVariablesDataRep::variablesView.

13.193.3 Member Data Documentation

### SizetArray variablesCompsTotals [private]

totals for variable type counts for \{continuous,discrete integer,discrete string,discrete real\} \{design,aleatory uncertain,epistemic uncertain,state\}.

This data reflects the variable counts as originally specified and is not altered by relaxation.

Referenced by SharedVariablesDataRep::aleatory_uncertain_counts(), SharedVariablesDataRep::all_counts(), SharedVariablesDataRep::components_to_totals(), SharedVariablesDataRep::copy_rep(), SharedVariablesDataRep::design_counts(), SharedVariablesDataRep::epistemic_uncertain_counts(), SharedVariablesDataRep::initialize_active_components(), SharedVariablesDataRep::initialize_all_ids(), SharedVariablesDataRep::initialize_components_totals(), SharedVariablesDataRep::initialize_inactive_components(), SharedVariablesDataRep::relax_noncategorical(), SharedVariablesDataRep::state_counts(), SharedVariablesDataRep::uncertain_counts(), and SharedVariablesDataRep::view_start_counts().

### SizetArray activeVarsCompsTotals [private]

totals for active variable type counts for \{continuous,discrete integer,discrete string,discrete real\} \{design,aleatory uncertain,epistemic uncertain,state\}.

This data reflects the variable counts as originally specified and is not altered by relaxation.

Referenced by SharedVariablesDataRep::copy_rep(), and SharedVariablesDataRep::initialize_active_components().

### SizetArray inactiveVarsCompsTotals [private]

totals for inactive variable type counts for \{continuous,discrete integer,discrete string,discrete real\} \{design,aleatory uncertain,epistemic uncertain,state\}.

This data reflects the variable counts as originally specified and is not altered by relaxation.

Referenced by SharedVariablesDataRep::copy_rep(), and SharedVariablesDataRep::initialize_inactive_components().
SizetMultiArray allContinuousIds  [private]
array of 1-based position identifiers for the all continuous variables array
These identifiers define positions of the all continuous variables array within the total variable sequence. A primary use case is for defining derivative ids (DVV) based on an active subset.
Referenced by SharedVariablesDataRep::copy_rep(), and SharedVariablesDataRep::initialize_all_ids().
The documentation for this class was generated from the following files:
- SharedVariablesData.hpp
- SharedVariablesData.cpp

13.194 SimulationModel Class Reference
Derived model class which utilizes a simulation-based application interface to map variables into responses.
Inheritance diagram for SimulationModel:

```
Model
    |
    V
SimulationModel
```

Public Member Functions
- **SimulationModel (ProblemDescDB &problem_db)**
  constructor
- **~SimulationModel ()**
  destructor

Protected Member Functions
- **Interface & derived_interface ()**
  return userDefinedInterface
- **size_t solution_levels (bool lwr_bnd=true) const**
  return size of solnControlCostMap, optionally enforcing lower bound of 1 solution level
- **void solution_level_index (size_t lev_index)**
  activate entry in solnControlCostMap
- **size_t solution_level_index () const**
  return active entry in solnControlCostMap
- **RealVector solution_level_costs () const**
  return all cost estimates from solnControlCostMap
- **Real solution_level_cost () const**
  return active cost estimate from solnControlCostMap
- **void derived_evaluate (const ActiveSet &set)**
  portion of evaluate() specific to SimulationModel (invokes a synchronous map() on userDefinedInterface)
- **void derived_evaluate_nowait (const ActiveSet &set)**
portions of `evaluate_nowait()` specific to `SimulationModel` (invokes an asynchronous map() on `userDefinedInterface`)

- `const IntResponseMap & derived_synchronize ()`  
  portion of `synchronize()` specific to `SimulationModel` (invokes `synch()` on `userDefinedInterface`)

- `const IntResponseMap & derived_synchronize_nowait ()`  
  portion of `synchronize_nowait()` specific to `SimulationModel` (invokes `synch_nowait()` on `userDefinedInterface`)

- `void component_parallel_mode (short mode)` (short mode)  
  `SimulationModel` only supports parallelism in `userDefinedInterface`, so this virtual function redefinition is simply a sanity check.

- `short local_eval_synchronization ()`  
  return `userDefinedInterface` synchronization setting

- `int local_eval_concurrency ()`  
  return `userDefinedInterface` asynchronous evaluation concurrency

- `bool derived_master_overload ()` const  
  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to `userDefinedInterface`)

- `IntIntPair estimate_partition_bounds (int max_eval_concurrency)`  
  estimate the minimum and maximum partition sizes that can be utilized by this `Model`

- `void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)`  
  set up `SimulationModel` for parallel operations (request forwarded to `userDefinedInterface`)

- `void derived_init_serial ()`  
  set up `SimulationModel` for serial operations (request forwarded to `userDefinedInterface`).

- `void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)`  
  set active parallel configuration for the `SimulationModel` (request forwarded to `userDefinedInterface`)

- `void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)`  
  Service `userDefinedInterface` job requests received from the master. Completes when a termination message is received from `stop_servers()`.

- `void stop_servers ()`  
  executed by the master to terminate `userDefinedInterface` server operations when `SimulationModel` iteration is complete.

- `const String & interface_id ()` const  
  return the `userDefinedInterface` identifier

- `int derived_evaluation_id ()` const  
  return the current evaluation id (`simModelEvalCnt`).

- `bool evaluation_cache (bool recurse_flag=true)` const  
  return flag indicated usage of an evaluation cache by the `SimulationModel` (request forwarded to `userDefinedInterface`)

- `bool restart_file (bool recurse_flag=true)` const  
  return flag indicated usage of a restart file by the `SimulationModel` (request forwarded to `userDefinedInterface`)

- `void set_evaluation_reference ()`  
  set the evaluation counter reference points for the `SimulationModel` (request forwarded to `userDefinedInterface`)

- `void fine_grained_evaluation_counters ()`  
  request fine-grained evaluation reporting within the `userDefinedInterface`

- `void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true)` const
print the evaluation summary for the SimulationModel (request forwarded to userDefinedInterface)

- void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix

Private Member Functions

- void initialize_solution_control (const String &control, const RealVector &cost)
  process the solution level inputs to define solnControlVarIndex, solnControlVarType, and solnControlCostMap

Private Attributes

- Interface userDefinedInterface
  the interface used for mapping variables to responses
- short solnCntlVarType
  type of the discrete variable that controls the set/range of solution levels
- size_t solnCntlADVIndex
  index of the discrete variable (within all view) that controls the set/range of solution levels
- size_t solnCntlSetIndex
  index of the discrete set variable (within its type array, managing offset when solnCntlVarType is a subset of all
discrete variables) that controls the set/range of solution levels
- std::map< Real, size_t > solnCntlCostMap
  sorted array of relative costs associated with a set of solution levels
- size_t simModelEvalCntr
  counter for calls to derived_evaluate()/derived_evaluate_nowait()
- IntIntMap simIdMap
  map from userDefinedInterface evaluation ids to SimulationModel ids (may differ in case where the same interface
instance is shared by multiple models)
- IntResponseMap simResponseMap
  map of simulation-based responses returned by derived_synchronize() and derived_synchronize_nowait()

Additional Inherited Members

13.194.1 Detailed Description

Derived model class which utilizes a simulation-based application interface to map variables into responses.
The SimulationModel class is the simplest of the derived model classes. It provides the capabilities of the
original Model class, prior to the development of surrogate and nested model extensions. The derived response
computation and synchronization functions utilize an application interface to perform the function evaluations.

13.194.2 Member Function Documentation

void eval_tag_prefix ( const String & eval_id_str ) [protected], [virtual]
set the hierarchical eval ID tag prefix
  SimulationModel doesn’t need to change the tagging, so just forward to Interface
  Reimplemented from Model.
  References Interface::eval_tag_prefix(), and SimulationModel::userDefinedInterface.
  The documentation for this class was generated from the following files:
13.195 SimulationResponse Class Reference

Container class for response functions and their derivatives. SimulationResponse provides the body class. Inheritance diagram for SimulationResponse:

```
Response
    |
    v
SimulationResponse
```

Public Member Functions

- **SimulationResponse ()**
  *default constructor*

- **SimulationResponse (const Variables &vars, const ProblemDescDB &problem_db)**
  *standard constructor built from problem description database*

- **SimulationResponse (const SharedResponseData &srd, const ActiveSet &set)**
  *alternate constructor that shares a SharedResponseData instance*

- **SimulationResponse (const SharedResponseData &srd)**
  *alternate constructor that shares a SharedResponseData instance*

- **SimulationResponse (const ActiveSet &set)**
  *alternate constructor using limited data*

- **∼SimulationResponse ()**
  *destructor*

Additional Inherited Members

13.195.1 Detailed Description

Container class for response functions and their derivatives. SimulationResponse provides the body class.

The SimulationResponse class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (Response) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (SimulationResponse) actually contains the response data (functionValues, function-Gradients, functionHessians, etc.). The representation is hidden in that an instance of SimulationResponse may only be created by Response. Therefore, programmers create instances of the Response handle class, and only need to be aware of the handle/body mechanisms when it comes to managing shallow copies (shared representation) versus deep copies (separate representation used for history mechanisms).

The documentation for this class was generated from the following files:

- SimulationResponse.hpp
- SimulationResponse.cpp
### 13.196 SNLLBase Class Reference

Base class for OPT++ optimization and least squares methods.
Inherited by SNLLLeastSq and SNLLOptimizer.

#### Public Member Functions

- **SNLLBase ()**
  default constructor
- **SNLLBase (ProblemDescDB &problem_db)**
  standard constructor
- **∼SNLLBase ()**
  destructor

#### Protected Member Functions

- **void copy_con_vals_dak_to_optpp (const RealVector &local_fn_vals, RealVector &g, size_t offset)**
  convenience function for copying local fn vals to g; used by constraint evaluator functions
- **void copy_con_vals_optpp_to_dak (const RealVector &g, RealVector &local_fn_vals, size_t offset)**
  convenience function for copying g to local_fn_vals; used in final solution logging
- **void copy_con_grad (const RealMatrix &local_fn_grads, RealMatrix &grad_g, size_t offset)**
  convenience function for copying local fn grads to grad g; used by constraint evaluator functions
- **void copy_con_hess (const RealSymMatrixArray &local_fn_hessians, OPTPP::OptppArray<RealSymMatrix> &hess_g, size_t offset)**
  convenience function for copying local fn hessians to hess g; used by constraint evaluator functions
- **void snll_pre_instantiate (bool bound_constr_flag, int num_constr)**
  convenience function for setting OPT++ options prior to the method instantiation
- **void snll_post_instantiate (int num_cv, bool vendor_num_grad_flag, const String &finite_diff_type, const RealVector &fss, int max_iter, int max_fn_evals, Real conv_tol, Real grad_tol, Real max_step, bool bound_constr_flag, int num_constr, short output_lev, OPTPP::OptimizeClass *theOptimizer, OPTPP::NLP0 *nlf_objective, OPTPP::FDNLF1 *fd_nlf1, OPTPP::FDNLF1 *fd_nlf1_con)**
  convenience function for setting OPT++ options after the method instantiation
- **void snll_initialize_run (OPTPP::NLP0 *nlf_objective, OPTPP::NLP *nlp_constraint, const RealVector &init_pt, bool bound_constr_flag, const RealVector &lower_bnds, const RealVector &upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nln_ineq_u_bnds, const RealVector &nln_ineq_u_l_bnds, const RealVector &nln_eq_targets)**
  convenience function for OPT++ configuration prior to the method invocation
- **void snll_post_run (OPTPP::NLP0 *nlf_objective)**
  convenience function for setting OPT++ options after the method instantiations

#### Static Protected Member Functions

- **static void init_fn (int n, RealVector &x)**
  An initialization mechanism provided by OPT++ (not currently used).
CHAPTER 13. CLASS DOCUMENTATION

Protected Attributes

- String `searchMethod`
  
  `value_based_line_search`, `gradient_based_line_search`, `trust_region`, or `tr_pds`

- OPTPP::SearchStrategy `searchStrat`
  
  `enum`: `LineSearch`, `TrustRegion`, or `TrustPDS`

- OPTPP::MeritFcn `meritFn`
  
  `enum`: `NormFmu`, `ArgaezTapia`, or `VanShanno`

- Real `maxStep`
  
  `value from max_step specification`

- Real `stepLenToBndry`
  
  `value from steplength_to_boundary specification`

- Real `centeringParam`
  
  `value from centering parameter specification`

- bool `constantASVFlag`
  
  `flags a user selection of active_set_vector == constant. By mapping this into mode override, reliance on duplicate detection can be avoided.`

Static Protected Attributes

- static `Minimizer * optLSqInstance`
  
  `pointer to the active base class object instance used within the static evaluator functions in order to avoid the need for static data`

- static bool `modeOverrideFlag`
  
  `flags OPT++ mode override (for combining value, gradient, and Hessian requests)`

- static EvalType `lastFnEvalLocn`
  
  `an enum used to track whether an nlf evaluator or a constraint evaluator was the last location of a function evaluation`

- static int `lastEvalMode`
  
  `copy of mode from constraint evaluators`

- static RealVector `lastEvalVars`
  
  `copy of variables from constraint evaluators`

13.196.1 Detailed Description

Base class for OPT++ optimization and least squares methods.

The `SNLLBase` class provides a common base class for SNLLOptimizer and SNLLLeastSq, both of which are wrappers for OPT++, a C++ optimization library from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site.

The documentation for this class was generated from the following files:

- `SNLLBase.hpp`
- `SNLLBase.cpp`
13.197 SNLLLeastSqTraits Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLLeastSqTraits:

```
TraitsBase
```

Public Member Functions

- **SNLLLeastSqTraits ()**
  
  *default constructor*

- **virtual ~SNLLLeastSqTraits ()**
  
  *destructor*

- **virtual bool is-derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality ()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

13.197.1 Detailed Description

Wrapper class for the OPT++ optimization library.

The SNLLLeastSq class provides a wrapper for OPT++, a C++ optimization library of nonlinear programming and pattern search techniques from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function, a static member, or accessed by static pointer.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, max_step, gradient_tolerance, search_method, and search_scheme_size are set using OPT++’s setMaxIter(), setMaxFeval(), setFcnTol(), setMaxStep(), setGradTol(), setSearchStrategy(), and setSSS() member functions, respectively; output verbosity is used to toggle OPT++’s debug mode using the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOTA search_method specification supports 4 (value_based_line_search, gradient_based_line_search, trust_region, or tr_pds). The difference stems from the "is_expensive" flag in OPT++. If the search strategy is LineSearch and "is_expensive" is turned on, then the value_based_line_search is used.
Otherwise (the “is_expensive” default is off), the algorithm will use the gradient_based_line_search. Refer to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/packages/OPTPP directory for information on OPT++ class member functions. A version of TraitsBase specialized for SNLLLeastSq

The documentation for this class was generated from the following file:

- SNLLLeastSq.hpp

### 13.198 SNLLTraits Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLTraits:

```
TraitsBase
   -- SNLLTraits
```

#### Public Member Functions

- **SNLLTraits ()**
  
  default constructor

- **virtual ~SNLLTraits ()**
  
  destructor

- **virtual bool is Derived ()**
  
  A temporary query used in the refactor.

- **bool supports_continuous_variables ()**
  
  Return the value of supportsContinuousVariables.

- **bool supports_linear_equality ()**
  
  Return the flag indicating whether method supports linear equalities.

- **bool supports_linear_inequality ()**
  
  Return the flag indicating whether method supports linear inequalities.

- **bool supports_nonlinear_equality ()**
  
  Return the flag indicating whether method supports nonlinear equalities.

- **bool supports_nonlinear_inequality ()**
  
  Return the flag indicating whether method supports nonlinear inequalities.

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  Return the format used for nonlinear inequality constraints.

#### 13.198.1 Detailed Description

Wrapper class for the OPT++ optimization library.

The SNLLOptimizer class provides a wrapper for OPT++, a C++ optimization library of nonlinear programming and pattern search techniques from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. It uses a function pointer approach for which passed functions must be either
global functions or static member functions. Any attribute used within static member functions must be either
to that function, a static member, or accessed by static pointer.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence-
tolerance, max_step, gradient_tolerance, search_method, and search_scheme_size are
set using OPT++'s setMaxIter(), setMaxFeval(), setFcnTol(), setMaxStep(), setGradTol(), setSearchStrategy(),
and setSSS() member functions, respectively; output verbosity is used to toggle OPT++'s debug mode using
the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOT-
A search_method specification supports 4 (value_based_line_search, gradient_based_line-
search, trust_region, or tr_pds). The difference stems from the ”is_expensive” flag in OPT++. If the
search strategy is LineSearch and ”is_expensive” is turned on, then the value_based_line_search is used.
Otherwise (the ”is_expensive” default is off), the algorithm will use the gradient_based_line_search. Ref-

to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/packages/OPTPP directory for information on
OPT++ class member functions. A version of TraitsBase specialized for SNLL optimizers

The documentation for this class was generated from the following file:

• SNLLOptimizer.hpp

13.199 SOLBase Class Reference

Base class for Stanford SOL software.
Inherited by NLSSOLLeastSq, and NPSOLOptimizer.

Public Member Functions

• SOLBase ()
  default constructor
• SOLBase (Model &model)
  standard constructor
• ~SOLBase ()
  destructor

Protected Member Functions

• void allocate_arrays (int num_cv, size_t num_nln_con, const RealMatrix &lin_ineq_coeffs, const RealMatrix
  &lin_eq_coeffs)
  Allocates miscellaneous arrays for the SOL algorithms.
• void deallocate_arrays ()
  Deallocates memory previously allocated by allocate_arrays().
• void allocate_workspace (int num_cv, int num_nln_con, int num_lin_con, int num_lsq)
  Allocates real and integer workspaces for the SOL algorithms.
• void set_options (bool speculative_flag, bool vendor_num_grad_flag, short output_lev, int verify_lev, Real
  fn_prec, Real linesrch_tol, int max_iter, Real constr_tol, Real conv_tol, const std::string &grad_type, const
  RealVector &fdss)
  Sets SOL method options using calls to npoptn2.
• void augment_bounds (RealVector &augmented_l_bnds, RealVector &augmented_u_bnds, const Model &model)
  augments variable bounds with linear and nonlinear constraint bounds.
• void augment bounds (RealVector &augmented_l_bnds, RealVector &augmented_u_bnds, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealVector &lin_eq_targets, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_targets)

  augments variable bounds with linear and nonlinear constraint bounds.

Static Protected Member Functions

• static void constraint eval (int &mode, int &ncnln, int &n, int &nrowj, int *needc, double *x, double *c, double *cjac, int &nstate)

  CONFUN in NPSOL manual: computes the values and first derivatives of the nonlinear constraint functions.

Protected Attributes

• int realWorkSpaceSize
  size of realWorkSpace

• int intWorkSpaceSize
  size of intWorkSpace

• RealArray realWorkSpace
  real work space for NPSOL/NLSSOL

• IntArray intWorkSpace
  int work space for NPSOL/NLSSOL

• int nlnConstraintArraySize
  used for non-zero array sizing (nonlinear constraints)

• int linConstraintArraySize
  used for non-zero array sizing (linear constraints)

• RealArray cLambda
  CLAMBDA from NPSOL manual: Langrange multipliers.

• IntArray constraintState
  ISTATE from NPSOL manual: constraint status.

• int informResult
  INFORM from NPSOL manual: optimization status on exit.

• int numberIterations
  ITER from NPSOL manual: number of (major) iterations performed.

• int boundsArraySize
  length of augmented bounds arrays (variable bounds plus linear and nonlinear constraint bounds)

• double * linConstraintMatrixF77
  [A] matrix from NPSOL manual: linear constraint coefficients

• double * upperFactorHessianF77

• double * constraintJacMatrixF77
  [CJAC] matrix from NPSOL manual: nonlinear constraint Jacobian

• int fnEvalCnt
  counter for testing against maxFunctionEvals

• size_t constrOffset
  used in constraint eval() to bridge NLSSOLLeastSq::numLeastSqTerms and NPSOLOptimizer::numObjectiveFns
Static Protected Attributes

- static SOLBase * solInstance
  
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

- static Minimizer * optLSqInstance
  
  pointer to the active base class object instance used within the static evaluator functions in order to avoid the need for static data

13.199.1 Detailed Description

Base class for Stanford SOL software.

The SOLBase class provides a common base class for NPSOLOptimizer and NLSSOLLeastSq, both of which are Fortran 77 sequential quadratic programming algorithms from Stanford University marketed by Stanford Business Associates.

The documentation for this class was generated from the following files:

- SOLBase.hpp
- SOLBase.cpp

13.200 SpawnApplicInterface Class Reference

Derived application interface class which spawns simulation codes using spawnvp.

Inheritance diagram for SpawnApplicInterface:

```
  Interface
    
  ApplicationInterface
    
  ProcessApplicInterface
    
  ProcessHandleApplicInterface
    
  SpawnApplicInterface
```

Public Member Functions

- SpawnApplicInterface (const ProblemDescDB &problem_db)
  
  constructor

- ~SpawnApplicInterface ()
  
  destructor
Protected Member Functions

- void wait_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- void test_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- pid_t create_analysis_process (bool block_flag, bool new_group)
  spawn a child process for an analysis component within an evaluation

- size_t wait_local_analyses ()
  wait for asynchronous analyses on the local processor, completing at least one job

- size_t test_local_analyses_send (int analysis_id)
  test for asynchronous analysis completions on the local processor and return results for any completions by sending messages

Additional Inherited Members

13.200.1 Detailed Description

Derived application interface class which spawns simulation codes using spawnvp. SpawnApplicInterface is used on Windows systems and is a peer to ForkApplicInterface for Unix systems. The documentation for this class was generated from the following files:

- SpawnApplicInterface.hpp
- SpawnApplicInterface.cpp

13.201 SurfpackApproximation Class Reference

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota. Inheritance diagram for SurfpackApproximation:

```
+------------------------
|                       |
|  Approximation         |
|  +---------------------
|  | SurfpackApproximation|
|  +---------------------
```

Public Member Functions

- SurfpackApproximation ()
  default constructor

- SurfpackApproximation (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)
  standard constructor: Surfpack surface of appropriate type will be created

- SurfpackApproximation (const SharedApproxData &shared_data)
Protected Member Functions

- int min_coefficients () const
  
  return the minimum number of samples (unknowns) required to build the derived class approximation type in numVars dimensions

- int recommended_coefficients () const
  
  return the recommended number of samples (unknowns) required to build the derived class approximation type in numVars dimensions

- void build (size_t index=NPOS)
  
  SurfData object will be created from Dakota’s SurrogateData, and the appropriate Surfpack build method will be invoked.

- void export_model (const String &fn_label, const String &export_prefix="", const unsigned short export_format=NO_MODEL_FORMAT)
  
  export the Surfpack model to disk or console

- Real value (const Variables &vars)
  
  Return the value of the Surfpack surface for a given parameter vector x.

- const RealVector & gradient (const Variables &vars)
  
  retrieve the approximate function gradient for a given parameter vector x

- const RealSymMatrix & hessian (const Variables &vars)
  
  retrieve the approximate function Hessian for a given parameter vector x

- Real prediction_variance (const Variables &vars)
  
  retrieve the variance of the predicted value for a given parameter set x (KrigingModel only)

- Real value (const RealVector &c_vars)
  
  Return the value of the Surfpack surface for a given parameter vector x.

- const RealVector & gradient (const RealVector &c_vars)
  
  retrieve the approximate function gradient for a given parameter vector x

- const RealSymMatrix & hessian (const RealVector &c_vars)
  
  retrieve the approximate function Hessian for a given parameter vector x

- Real prediction_variance (const RealVector &c_vars)
  
  retrieve the variance of the predicted value for a given parameter set x (KrigingModel only)

- bool diagnostics_available ()
  
  check if the diagnostics are available (true for the Surfpack types)

- Real diagnostic (const String &metric_type)
  
  retrieve a single diagnostic metric for the diagnostic type specified on the primary model and data

- Real diagnostic (const String &metric_type, const SurfpackModel &model, const SurfData &data)
  
  retrieve a single diagnostic metric for the diagnostic type specified on the given model and data

- void primary_diagnostics (int fn_index)
  
  compute and print all requested diagnostics and cross-validation

- void challenge_diagnostics (int fn_index, const RealMatrix &challenge_points, const RealVector &challenge_responses)
CHAPTER 13. CLASS DOCUMENTATION

computes and prints all requested diagnostics for user-provided challenge points

- **RealArray cv_diagnostic** (const StringArray &metric_types, unsigned num_folds)
  compute and return cross-validation for metric type with num_folds

- **RealArray challenge_diagnostic** (const StringArray &metric_types, const RealMatrix &challenge_points, const RealVector &challenge_responses)
  compute and print all requested diagnostics for user-provided challenge points

**Private Member Functions**

- **SurfData * surrogates_to_surf_data ()**
  copy from SurrogateData to SurfPoint/SurfData

- **void add_anchor_to_surfdata (SurfData &surf_data)**
  set the anchor point (including gradient and hessian if present) into surf_data

**Private Attributes**

- **SurfpackModel * model**
  The native Surfpack approximation.

- **SurfpackModelFactory * factory**
  factory for the SurfpackModel instance

- **SurfData * surfData**
  The data used to build the approximation, in Surfpack format.

**Additional Inherited Members**

13.201.1 Detailed Description

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

The SurfpackApproximation class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SurfpackApproximation builds a Surfpack Surface object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial neural networks, radial basis function network, or multivariate adaptive regression splines (MARS).

13.201.2 Constructor & Destructor Documentation

SurfpackApproximation (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)

standard constructor: Surfpack surface of appropriate type will be created.

Initialize the embedded Surfpack surface object and configure it using the specifications from the input file. Data for the surface is created later.

References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, SharedApproxData::approxType, SharedApproxData::buildDataOrder, Dakota::copy_data(), SharedSurfpackApproxData::crossValidateFlag, SharedSurfpackApproxData::diagnosticSet, SurfpackApproximation::factory, ProblemDescDB::get_real(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), SharedSurfpackApproxData::numFolds, SharedApproxData::numVars, SharedApproxData::outputLevel, SharedSurfpackApproxData::percentFold, and Approximation::sharedDataRep.
13.201.1  

**SurfpackApproximation** ( const SharedApproxData & shared_data )

alternate constructor

On-the-fly constructor which uses mostly Surfpack model defaults.

References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, SharedApproxData::approxType, SharedApproxData::buildDataOrder, SurfpackApproximation::factory, SharedApproxData::numVars, SharedApproxData::outputLevel, and Approximation::sharedDataRep.

13.201.3  Member Function Documentation

**void build ( size_t index = _NPOS )** [protected], [virtual]

SurfData object will be created from Dakota’s SurrogateData, and the appropriate Surfpack build method will be invoked.

surfData will be deleted in dtor

**Todo** Right now, we’re completely deleting the old data and then recopying the current data into a SurfData object. This was just the easiest way to arrive at a solution that would build and run. This function is frequently called from addPoint rebuild, however, and it’s not good to go through this whole process every time one more data point is added.

Reimplemented from Approximation.

References Dakota::abort_handler(), SharedApproxData::approxCLowerBnds, SharedApproxData::approxCUpperBnds, SharedApproxData::approxDILowerBnds, SharedApproxData::approxDIUpperBnds, SharedApproxData::approxDRLowerBnds, SharedApproxData::approxDRUpperBnds, Approximation::build(), SurfpackApproximation::factory, SharedSurfpackApproxData::merge_variable_arrays(), SurfpackApproximation::model, Approximation::sharedDataRep, SurfpackApproximation::surfData, and SurfpackApproximation::surrogates_to_surf_data().

**const RealSymMatrix & hessian ( const Variables & vars )** [protected], [virtual]

retrieve the approximate function Hessian for a given parameter vector x

**Todo** Make this acceptably efficient

Reimplemented from Approximation.

References Dakota::abort_handler(), Approximation::approxHessian, SharedApproxData::approxType, Variables::cv(), SurfpackApproximation::model, and Approximation::sharedDataRep.

Referenced by SurfpackApproximation::add_anchor_to_surfdata().

**const RealSymMatrix & hessian ( const RealVector & c_vars )** [protected], [virtual]

retrieve the approximate function Hessian for a given parameter vector x

**Todo** Make this acceptably efficient

Reimplemented from Approximation.

References Dakota::abort_handler(), Approximation::approxHessian, SharedApproxData::approxType, SurfpackApproximation::model, and Approximation::sharedDataRep.
SurfData * surrogates_to_surf_data( ) [private]

copy from SurrogateData to SurfPoint/SurfData
Copy the data stored in Dakota-style SurrogateData into Surfpack-style SurfPoint and SurfData objects.
References SurfpackApproximation::add_anchor_to_surfdata(), SharedSurfpackApproxData::add_sd_to_surfdata(),
Approximation::approxData, SharedApproxData::buildDataOrder, SurfpackApproximation::factory, SharedApprox-
Data::outputLevel, and Approximation::sharedDataRep.
Referenced by SurfpackApproximation::build().

void add_anchor_to_surfdata( SurfData & surfdata ) [private]

set the anchor point (including gradient and hessian if present) into surfdata
If there is an anchor point, add an equality constraint for its response value. Also add constraints for gradient
and hessian, if applicable.
References Dakota::abort_handler(), Approximation::approxData, Dakota::copy_data(), SharedSurfpackApprox-
Data::copy_matrix(), SurfpackApproximation::gradient(), SurfpackApproximation::hessian(), SharedApproxData-
::outputLevel, SharedSurfpackApproxData::sdv_to_realarray(), and Approximation::sharedDataRep.
Referenced by SurfpackApproximation::surrogates_to_surf_data().
The documentation for this class was generated from the following files:
• SurfpackApproximation.hpp
• SurfpackApproximation.cpp

13.202 SurrBasedGlobalMinimizer Class Reference

Inheritance diagram for SurrBasedGlobalMinimizer:

```
  Iterator
   |
   Minimizer
   |
SurrBasedMinimizer
   |
SurrBasedGlobalMinimizer
```

Public Member Functions

- SurrBasedGlobalMinimizer (ProblemDescDB &problem_db, Model &model)
  constructor
- ~SurrBasedGlobalMinimizer ()
  destructor
Protected Member Functions

- void core_run ()

Performing global surrogate-based optimization by repeatedly optimizing on and improving surrogates of the response functions.

- bool returns_multiple_points () const

Global surrogate-based methods can return multiple points.

Private Attributes

- bool replacePoints

Flag for replacing the previous iteration’s point additions, rather than continuing to append, during construction of the next surrogate.

Additional Inherited Members

13.202.1 Detailed Description

This method uses a SurrBasedGlobalMinimizer to perform minimization (optimization or nonlinear least squares) through a set of iterations. At each iteration, a surrogate is built, the surrogate is minimized, and the optimal points from the surrogate are then evaluated with the “true” function, to generate new points upon which the surrogate for the next iteration is built.

The documentation for this class was generated from the following files:

- SurrBasedGlobalMinimizer.hpp
- SurrBasedGlobalMinimizer.cpp

13.203 SurrBasedGlobalTraits Class Reference

The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls.

Inheritance diagram for SurrBasedGlobalTraits:

```
           TraitsBase
            |
  SurrBasedGlobalTraits
```

Public Member Functions

- SurrBasedGlobalTraits ()

Default constructor

- virtual ~SurrBasedGlobalTraits ()

Destructor

- virtual bool is_derived ()

A temporary query used in the refactoring.

- bool supports_continuous_variables ()
Return the flag indicating whether method supports continuous variables.

- bool supports_discrete_variables ()
  Return the flag indicating whether method supports discrete variables.

- bool supports_linear_equality ()
  Return the flag indicating whether method supports linear equalities.

- bool supports_linear_inequality ()
  Return the flag indicating whether method supports linear inequalities.

- bool supports_nonlinear_equality ()
  Return the flag indicating whether method supports nonlinear equalities.

- bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.

### 13.203.1 Detailed Description

The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls.

A version of TraitsBase specialized for surrogate-based global minimizer

The documentation for this class was generated from the following file:

- SurrBasedGlobalMinimizer.hpp

### 13.204 SurrBasedLocalMinimizer Class Reference

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

Inheritance diagram for SurrBasedLocalMinimizer:

```
  Iterator
    Minimizer
      SurrBasedMinimizer
        SurrBasedLocalMinimizer
```

### Public Member Functions

- SurrBasedLocalMinimizer (ProblemDescDB &problem_db, Model &model, std::shared_ptr<TraitsBase> traits)
  constructor

- ~SurrBasedLocalMinimizer ()
  destructor
Protected Member Functions

- void **pre_run** ()
  
  *pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- void **core_run** ()
  
  *Performs local surrogate-based minimizing by minimizing local, global, or hierarchical surrogates over a series of trust regions.*

- void **post_run** (std::ostream &s)
  
  *post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way*

- void **reset** ()
  
  *restore initial state for repeated sub-iterator executions*

- virtual **SurrBasedLevelData & trust_region** ()=0
  
  *return the active SurrBasedLevelData instance*

- virtual void **update_trust_region** ()=0
  
  *update the trust region bounds, strictly contained within global bounds*

- virtual void **build** ()=0
  
  *build the approximation over the current trust region*

- virtual void **minimize** ()
  
  *solve the approximate subproblem*

- virtual void **verify** ()=0
  
  *verify the approximate iterate and update the trust region for the next approximate optimization cycle*

- virtual unsigned short **converged** ()=0
  
  *return the convergence code for the truth level of the trust region hierarchy*

- void **initialize_sub_model** ()
  
  *construct and initialize approxSubProbModel*

- void **initialize_sub_minimizer** ()
  
  *construct and initialize approxSubProbMinimizer*

- void **initialize_multipliers** ()
  
  *initialize lagrangeMult and augLagrangeMult*

- void **reset_penalties** ()
  
  *reset all penalty parameters to their initial values*

- void **reset_multipliers** ()
  
  *reset Lagrange multipliers to initial values for cases where they are accumulated instead of computed directly*

- void **update_trust_region_data** (SurrBasedLevelData &tr_data, const RealVector &parent_l_bnds, const RealVector &parent_u_bnds)
  
  *update the trust region bounds, strictly contained within global bounds*

- void **update_approx_sub_problem** (SurrBasedLevelData &tr_data)
  
  *update variables and bounds within approxSubProbModel*

- void **compute_trust_region_ratio** (SurrBasedLevelData &tr_data, bool check_interior=false)
  
  *compute trust region ratio (for SBLM iterate acceptance and trust region resizing) and check for soft convergence (diminishing returns)*

- void **hard_convergence_check** (SurrBasedLevelData &tr_data, const RealVector &lower_bnds, const RealVector &upper_bnds)
check for hard convergence (norm of projected gradient of merit function < tolerance)

- **void print_convergence_code** (std::ostream &s, unsigned short code)
  
  print out the state corresponding to the code returned by converged()

- **void update_penalty** (const RealVector &fns_center_truth, const RealVector &fns_star_truth)
  
  initialize and update the penalty Parameter

- **bool find_approx_response** (const Variables &search_vars, Response &search_resp)
  
  locate an approximate response with the data pairs cache

- **bool find_truth_response** (const Variables &search_vars, Response &search_resp)
  
  locate a truth response with the data pairs cache

- **bool find_response** (const Variables &search_vars, Response &search_resp, const String &search_id, short set_request)
  
  locate a response with the data pairs cache

- **void relax_constraints** (SurrBasedLevelData &tr_data)
  
  relax constraints by updating bounds when current iterate is infeasible

### Static Protected Member Functions

- **static void approx_subprob_objective_eval** (const Variables &surrogate_vars, const Variables &recast_vars, const Response &surrogate_response, Response &recast_response)
  
  static function used to define the approximate subproblem objective.

- **static void approx_subprob_constraint_eval** (const Variables &surrogate_vars, const Variables &recast_vars, const Response &surrogate_response, Response &recast_response)
  
  static function used to define the approximate subproblem constraints.

- **static void hom_objective_eval** (int &mode, int &n, double *tau and x, double &f, double *grad_f, int &)
  
  static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.

- **static void hom_constraint_eval** (int &mode, int &ncnln, int &n, int &nrowj, int *neecd, double *tau and x, double *c, double *cjac, int &nstate)
  
  static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation.

### Protected Attributes

- **Model approxSubProbModel**
  
  the approximate sub-problem formulation solved on each approximate minimization cycle: may be a shallow copy of iteratedModel, or may involve a RecastModel recursion applied to iteratedModel

- **short approxSubProbObj**
  
  type of approximate subproblem objective: ORIGINAL_OBJ, LAGRANGIAN_OBJ, or AUGMENTED_LAGRANGIAN_OBJ

- **short approxSubProbCon**
  
  type of approximate subproblem constraints: NO_CON, LINEARIZED_CON, or ORIGINAL_CON

- **bool recastSubProb**
  
  flag to indicate when approxSubProbModel involves a RecastModel recursion

- **short meritFnType**
  
  type of merit function used in trust region ratio logic: PENALTY_MERIT, ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, or AUGMENTED_LAGRANGIAN_MERIT

- **short acceptLogic**
type of iterate acceptance test logic: FILTER or TR_RATIO

- short trConstraintRelax
  type of trust region constraint relaxation for infeasible starting points: NO_RELAX or HOMOTOPY

- int minimizeCycles
  counter for number of minimization cycles that have accumulated prior to convergence at the minimizeIndex level (used for ramping penalties)

- int penaltyIterOffset
  iteration offset used to update the scaling of the penalty parameter for adaptive penalty merit functions

- RealVector origTrustRegionFactor
  original user specification for trust region initial size

- Real minTrustRegionFactor
  a soft convergence control: stop SBLM when the trust region factor is reduced below the value of minTrustRegionFactor

- Real trRatioContractValue
  trust region ratio min value: contract tr if ratio below this value

- Real trRatioExpandValue
  trust region ratio sufficient value: expand tr if ratio above this value

- Real gammaContract
  trust region contraction factor

- Real gammaExpand
  trust region expansion factor

- unsigned short softConvLimit
  convergence control limiting the number of consecutive iterations that fail to achieve sufficient decrease. If exceeded by softConvCount, stop.

- short truthSetRequest
  derivative order of truth data used within the SBLM process

- short approxSetRequest
  derivative order of surrogate data used within the SBLM process

- short correctionType
  flags the use of surrogate correction techniques at the center of each trust region

- RealVector initialPoint
  starting point prior to sequence of SBLM iterations

- RealVector globalLowerBnds
  Global lower bounds.

- RealVector globalUpperBnds
  Global Upper bounds.

- RealVector nonlinIneqLowerBndsSlack
  individual violations of nonlinear inequality constraint lower bounds

- RealVector nonlinIneqUpperBndsSlack
  individual violations of nonlinear inequality constraint upper bounds

- RealVector nonlinEqTargetsSlack
  individual violations of nonlinear equality constraint targets

- Real tau
  constraint relaxation parameter

- Real alpha
  constraint relaxation parameter backoff parameter (multiplier)
Static Protected Attributes

- static SurrogateLocalMinimizer * sblmInstance
  pointer to SBLM instance used in static member functions

Additional Inherited Members

13.204.1 Detailed Description

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

This minimizer uses a SurrogateModel to perform minimization based on local, global, or hierarchical surrogates. It achieves provable convergence through the use of a sequence of trust regions and the application of surrogate corrections at the trust region centers.

13.204.2 Member Function Documentation

void pre_run ( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Iterator.

References Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), SurrogateLocalMinimizer::converged(), Dakota::copy_data(), SurrogateLocalMinimizer::globalLowerBnds, SurrogateLocalMinimizer::globalUpperBnds, SurrogateLocalMinimizer::initialPoint, Iterator::iteratedModel, and SurrogateLocalMinimizer::reset().

void core_run ( ) [protected], [virtual]

Performs local surrogate-based minimization by minimizing local, global, or hierarchical surrogates over a series of trust regions.

Trust region-based strategy to perform surrogate-based optimization in subregions (trust regions) of the parameter space. The minimizer operates on approximations in lieu of the more expensive simulation-based response functions. The size of the trust region is adapted according to the agreement between the approximations and the true response functions.

Reimplemented from Iterator.

References SurrogateLocalMinimizer::build(), SurrogateLocalMinimizer::converged(), SurrogateLocalMinimizer::minimize(), SurrogateLocalMinimizer::sblmInstance, SurrogateLocalMinimizer::update_trust_region(), and SurrogateLocalMinimizer::verify().

void post_run ( std::ostream & s ) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way.

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Minimizer.
References SurrBasedLocalMinimizer::converged(), SurrBasedMinimizer::globalIterCount, Minimizer::post-run(), and SurrBasedLocalMinimizer::print_convergence_code().

```cpp
void compute_trust_region_ratio ( SurrBasedLevelData & tr_data, bool check_interior = false )
[protected]
```
compute trust region ratio (for SBLM iterate acceptance and trust region resizing) and check for soft convergence (diminishing returns)

Assess acceptance of SBLM iterate (trust region ratio or filter) and compute soft convergence metrics (number of consecutive failures, min trust region size, etc.) to assess whether the convergence rate has decreased to a point where the process should be terminated (diminishing returns).

References SurrBasedLocalMinimizer::acceptLogic, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::augmented_lagrangian_merit(), SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, Iterator::convergenceTol, SurrBasedMinimizer::etaSequence, Response::function_values(), SurrBasedLocalMinimizer::gammaContract, SurrBasedLocalMinimizer::gammaExpand, SurrBasedMinimizer::initialize_filter(), Response::is_null(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_merit(), SurrBasedLocalMinimizer::meritFnType, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Iterator::outputLevel, SurrBasedMinimizer::penalty_merit(), SurrBasedMinimizer::penalty_merit(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Response::reset(), SurrBasedLocalMinimizer::trRatioContractValue, SurrBasedLocalMinimizer::trRatioExpandValue, SurrBasedMinimizer::update_augmented_lagrange_multipliers(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

```cpp
void hard_convergence_check ( SurrBasedLevelData & tr_data, const RealVector & lower_bnds, const RealVector & upper_bnds )
[protected]
```
check for hard convergence (norm of projected gradient of merit function < tolerance)

The hard convergence check computes the gradient of the merit function at the trust region center, performs a projection for active bound constraints (removing any gradient component directed into an active bound), and signals convergence if the 2-norm of this projected gradient is less than convergenceTol.

References SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, Iterator::convergenceTol, Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_gradient(), SurrBasedLocalMinimizer::meritFnType, SurrBasedLocalMinimizer::minimizeCycles, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Iterator::outputLevel, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), SurrBasedLocalMinimizer::truthSetRequest, SurrBasedMinimizer::update_augmented_lagrange_multipliers(), and SurrBasedMinimizer::update_lagrange_multipliers().

```cpp
void update_penalty ( const RealVector & fns_center_truth, const RealVector & fns_star_truth )
[protected]
```
initialize and update the penaltyParameter

Scaling of the penalty value is important to avoid rejecting SBLM iterates which must increase the objective to achieve a reduction in constraint violation. In the basic penalty case, the penalty is ramped exponentially based on the iteration counter. In the adaptive case, the ratio of relative change between center and star points for the objective and constraint violation values is used to rescale penalty values.

References SurrBasedMinimizer::alphaEta, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, SurrBasedMinimizer::eta, SurrBasedMinimizer::etaSequence,
void approx_subprob_objective_eval ( const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response ) [static], [protected]

static function used to define the approximate subproblem objective.

Objective functions evaluator for solution of approximate subproblem using a RecastModel.

References Response::active_set_request_vector(), SurrBasedLocalMinimizer::approxSubProbCon, SurrBasedLocalMinimizer::approxSubProbModel, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::augmented_lagrangian_gradient(), SurrBasedMinimizer::augmented_lagrangian_hessian(), SurrBasedMinimizer::augmented_lagrangian_merit(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessians(), Response::function_value(), Response::function_values(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_gradient(), SurrBasedMinimizer::lagrangian_hessian(), SurrBasedMinimizer::lagrangian_merit(), Model::numUserPrimaryFns, Minimizer::objective(), Minimizer::objective_gradient(), Minimizer::objective_hessian(), SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and SurrBasedLocalMinimizer::sblmInstance.

Referenced by SurrBasedLocalMinimizer::initialize_sub_model().

void approx_subprob_constraint_eval ( const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response ) [static], [protected]

static function used to define the approximate subproblem constraints.

Constraint functions evaluator for solution of approximate subproblem using a RecastModel.

References Response::active_set_derivative_vector(), Response::active_set_request_vector(), SurrBasedLocalMinimizer::approxSubProbCon, SurrBasedLocalMinimizer::approxSubProbModel, SurrBasedLocalMinimizer::approxSubProbObj, Variables::continuous_variables(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessians_view(), Response::function_gradients(), Response::function_value(), Response::function_values(), Minimizer::numUserPrimaryFns, SurrBasedLocalMinimizer::sblmInstance, and SurrBasedLocalMinimizer::trust_region().

Referenced by SurrBasedLocalMinimizer::initialize_sub_model().

void hom_objective_eval ( int & mode, int & n, double * tau_and_x, double & f, double * grad_f, int & ) [static], [protected]

static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.

NPSOL objective functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.

Referenced by SurrBasedLocalMinimizer::relax_constraints().

void hom_constraint_eval ( int & mode, int & ncnln, int & n, int & nrowj, int * needc, double * tau_and_x, double * c, double * cjac, int & nstate ) [static], [protected]

static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation.
NPSOL constraint functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.

References
- Response::active_set()
- SurrBasedLocalMinimizer::approxSubProbModel
- Model::continuous_variables()
- Model::current_response()
- Response::function_gradients()
- Response::function_values()
- SurrBasedLocalMinimizer::nonlinEqTargetsSlack
- SurrBasedLocalMinimizer::nonlinIneqUpperBndsSlack
- Model::num_functions()
- Minimizer::numNonlinearEqConstraints
- Minimizer::numNonlinearIneqConstraints
- ActiveSet::request_vector()
- SurrBasedLocalMinimizer::sblmInstance,
- and SurrBasedLocalMinimizer::tau.

Referenced by SurrBasedLocalMinimizer::relax_constraints().

The documentation for this class was generated from the following files:
- SurrBasedLocalMinimizer.hpp
- SurrBasedLocalMinimizer.cpp

### 13.205 SurrBasedMinimizer Class Reference

Base class for local/global surrogate-based optimization/least squares.

Inheritance diagram for SurrBasedMinimizer:

```
    Iterator
    ↓
  Minimizer
  ↓
SurrBasedMinimizer
  ↓
EffGlobalMinimizer  SurrBasedGlobalMinimizer  SurrBasedLocalMinimizer
```

**Protected Member Functions**

- **SurrBasedMinimizer** (ProblemDescDB &problem_db, Model &model, std::shared_ptr<TraitsBase> traits)
  - constructor
- ~SurrBasedMinimizer ()
  - destructor
- void derived_init_communicators (ParLevLIter pl_iter)
  - derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter)
  - derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter)
  - derived class contributions to freeing the communicators associated with this Iterator instance
- void initialize_graphics (int iterator_server_id=1)
  - initialize graphics customized for surrogate-based iteration
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
- void update_lagrange_multipliers (const RealVector &fn_vals, const RealMatrix &fn_grads)
initialize and update Lagrange multipliers for basic Lagrangian

- **void update_augmented_lagrange_multipliers** (const RealVector &fn_vals)
  *initialize and update the Lagrange multipliers for augmented Lagrangian*

- **void initialize_filter** (SurrBasedLevelData &tr_data, const RealVector &fn_vals)
  *(re-)initialize filter from a set of function values*

- **bool update_filter** (SurrBasedLevelData &tr_data, const RealVector &fn_vals)
  *update filter using a new set of function values*

- **Real lagrangian_merit** (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts)
  *compute a Lagrangian function from a set of function values*

- **void lagrangian_gradient** (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealVector &lag_grad)
  *compute the gradient of the Lagrangian function*

- **void lagrangian_hessian** (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealSymMatrix &lag_hess)
  *compute the Hessian of the Lagrangian function*

- **Real augmented_lagrange_merit** (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts)
  *compute an augmented Lagrangian function from a set of function values*

- **void augmented_lagrange_gradient** (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealVector &alag_grad)
  *compute the gradient of the augmented Lagrangian function*

- **void augmented_lagrange_hessian** (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealSymMatrix &alag_hess)
  *compute the Hessian of the augmented Lagrangian function*

- **Real penalty_merit** (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts)
  *compute a penalty function from a set of function values*

- **void penalty_gradient** (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, RealVector &pen_grad)
  *compute the gradient of the penalty function*

- **Real constraint_violation** (const RealVector &fn_vals, const Real &constraint_tol)
  *compute the constraint violation from a set of function values*

**Protected Attributes**

- **Iterator approxSubProbMinimizer**
  *the minimizer used on the surrogate model to solve the approximate subproblem on each surrogate-based iteration*
Additional Inherited Members

13.205.1 Detailed Description

Base class for local/global surrogate-based optimization/least squares.
These minimizers use a SurrogateModel to perform optimization based either on local trust region methods or global updating methods.

13.205.2 Member Function Documentation

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

Redefines default iterator results printing to include optimization results (objective functions and constraints).
Reimplemented from Iterator.

References Dakota::abort_handler(), Iterator::activeSet, Minimizer::archive_allocate_best(), Minimizer::archive_best(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Dakota::data_pairs, Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Iterator::methodName, Minimizer::numFunctions, Minimizer::numUserPrimaryFns, Minimizer::optimizationFlag, ActiveSet::request_values(), and Model::truth_model().
void update_lagrange_multipliers ( const RealVector & fn_vals, const RealMatrix & fn_grads )
[protected]
initialize and update Lagrange multipliers for basic Lagrangian

For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not
require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through
solution of a nonnegative linear least squares problem.

References Dakota::abort_handler(), Minimizer::bigRealBoundSize, Minimizer::constraintTol, Iterator::iterated-
Model, SurrBasedMinimizer::lagrangeMult, Minimizer::numContinuousVars, Minimizer::numNonlinearEqConstraints,
Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Minimizer::objective_gradient(), Surr-
BasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Model::primary-
_response_fn_sense(), and Model::primary_response_fn_weights().

Referenced by SurrBasedLocalMinimizer::hard_convergence_check().

void update_augmented_lagrange_multipliers ( const RealVector & fn_vals )
[protected]
initialize and update the Lagrange multipliers for augmented Lagrangian

For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not
require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through
solution of a nonnegative linear least squares problem.

References SurrBasedMinimizer::augLagrangeMult, SurrBasedMinimizer::betaEta, Minimizer::bigRealBound-
Size, SurrBasedMinimizer::etaSequence, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneq-
Constraints, Minimizer::numUserPrimaryFns, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer-
::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, and SurrBasedMinimizer::penalty-
Parameter.

Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio(), SurrBasedLocalMinimizer::hard_convergence-
_check(), and EffGlobalMinimizer::minimize_surrogates_on_model().

bool update_filter ( SurrBasedLevelData & tr_data, const RealVector & fn_vals )
[protected]
update filter using a new set of function values

Update the paretoFilter with fn_vals if new iterate is non-dominated.

References SurrBasedMinimizer::constraint_violation(), Iterator::iteratedModel, Minimizer::numNonlinear-
Constraints, Minimizer::objective(), Model::primary_response_fn_sense(), and Model::primary_response_fn_weights().

Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio().

Real lagrangian_merit ( const RealVector & fn_vals, const BoolDeque & sense, const RealVector &
primary_wts, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector
& nln_eq_tgts )
[protected]
compute a Lagrangian function from a set of function values

The Lagrangian function computation sums the objective function and the Lagrange multiplier terms for in-
equality/equality constraints. This implementation follows the convention in Vanderplaats with g<=0 and h=0.
The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

References Minimizer::bigRealBoundSize, Minimizer::constraintTol, SurrBasedMinimizer::lagrangeMult, Minimizer-
::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, and
Minimizer::objective().

Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), and SurrBasedLocalMinimizer-
::compute_trust_region_ratio().
Real augmented_lagrangian_merit ( const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts ) [protected]

compute an augmented Lagrangian function from a set of function values

The Rockafellar augmented Lagrangian function sums the objective function, Lagrange multiplier terms for inequality/equality constraints, and quadratic penalty terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with $g^<=0$ and $h=0$. The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

References SurrBasedMinimizer::augLagrangeMult, Minimizer::bigRealBoundSize, Minimizer::numNonlinear-EqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Minimizer::objective(), and SurrBasedMinimizer::penaltyParameter.

Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), SurrBasedLocalMinimizer::compute_trust_region_ratio(), EffGlobalMinimizer::get_best_sample(), and EffGlobalMinimizer::minimize_surrogates_on_model().

Real penalty_merit ( const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts ) [protected]

compute a penalty function from a set of function values

The penalty function computation applies a quadratic penalty to any constraint violations and adds this to the objective function(s) $p = f + r_p \cdot cv$.

References SurrBasedMinimizer::constraintViolation(), Minimizer::constraintTol, Minimizer::objective(), and SurrBasedMinimizer::penaltyParameter.

Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio().

Real constraint_violation ( const RealVector & fn_vals, const Real & constraint_tol ) [protected]

compute the constraint violation from a set of function values

Compute the quadratic constraint violation defined as $cv = g^+ \cdot h^+$.

- $h^+ \cdot h^+$. This implementation supports equality constraints and 2-sided inequalities. The constraint_tol allows for a small constraint infeasibility (used for penalty methods, but not Lagrangian methods).

References Minimizer::bigRealBoundSize, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, and SurrBasedMinimizer::origNonlinIneqUpperBnds.

Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio(), SurrBasedLocalMinimizer::hard_convergence_check(), SurrBasedMinimizer::initialize_filter(), EffGlobalMinimizer::minimize_surrogates_on_model(), SurrBasedMinimizer::penalty_merit(), SurrBasedLocalMinimizer::relax_constraints(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

The documentation for this class was generated from the following files:

- SurrBasedMinimizer.hpp
- SurrBasedMinimizer.cpp

13.206 SurrogateModel Class Reference

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).

Inheritance diagram for SurrogateModel:
Protected Member Functions

- **SurrogateModel (ProblemDescDB &problem_db)**
  constructor
- **SurrogateModel (ProblemDescDB &problem_db, ParallelLibrary &parallel_lib, const SharedVariablesData &svd, const SharedResponseData &srd, const ActiveSet &set, short corr_type, short output_level)**
  alternate constructor
- ~SurrogateModel ()
  destructor
- bool initialize_mapping (ParLevLIter pl_iter)
  initialize model mapping, returns true if the variables size has changed
- bool finalize_mapping ()
  finalize model mapping, returns true if the variables size has changed
- bool mapping_initialized () const
  return mappingInitialized
- Model & subordinate_model ()
  return truth_model()
- short surrogate_response_mode () const
  return responseMode
- int derived_evaluation_id () const
  return the current evaluation id for this Model
- size_t mi_parallel_level_index () const
  return miPLIndex
- void check_submodel_compatibility (const Model &sub_model)
  verify compatibility between SurrogateModel attributes and attributes of the submodel (DataFitSurrModel::actual-Model or HierarchSurrModel::highFidelityModel)
- void check_key (int key1, int key2) const
  check for consistency in response map keys
- bool force_rebuild ()
  evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data
- void asv_mapping (const ShortArray &orig_asv, ShortArray &actual_asv, ShortArray &approx_asv, bool build_flag)
  distributes the incoming orig_asv among actual_asv and approx_asv
- void asv_mapping (const ShortArray &actual_asv, const ShortArray &approx_asv, ShortArray &combined_asv)
  reconstitutes a combined_asv from actual_asv and approx_asv
• **void response_mapping** (const Response &actual_response, const Response &approx_response, Response &combined_response)
  overlays actual_response and approx_response to update combined_response

• **void aggregate_response** (const Response &hf_resp, const Response &lf_resp, Response &agg_resp)
  aggregate LF and HF response to create a new response with 2x size

**Protected Attributes**

• IntSet surrogateFnIndices
  for mixed response sets, this array specifies the response function subset that is approximated

• short responseMode
  an enumeration that controls the response calculation mode in {DataFit,Hierarch}SurrModel approximate response computations

• short corrType
  type of correction: additive, multiplicative, or combined

• IntIntMap truthIdMap
  map from actualModel/highFidelityModel evaluation ids to DataFitSurrModel/HierarchSurrModel ids

• IntIntMap surrIdMap
  map from approxInterface/lowFidelityModel evaluation ids to DataFitSurrModel/HierarchSurrModel ids

• int surrModelEvalCntnr
  counter for calls to derived_evaluate()/derived_evaluate_nowait(); used to key response maps from SurrogateModels

• IntResponseMap surrResponseMap
  map of surrogate responses returned by derived_synchronize() and derived_synchronize_nowait()

• IntResponseMap cachedApproxRespMap
  map of approximate responses retrieved in derived_synchronize_nowait() that could not be returned since corresponding truth model response portions were still pending.

• IntVariablesMap rawVarsMap
  map of raw continuous variables used by apply_correction(). Model::varsList cannot be used for this purpose since it does not contain lower level variables sets from finite differencing.

• size_t approxBuilds
  number of calls to build_approximation()

• size_t miPLIndex
  the index of the active metaiterator-iterator parallelism level (corresponding to ParallelConfiguration::miPLIters) used at runtime

• RealVector referenceCLBnds
  stores a reference copy of active continuous lower bounds when the approximation is built; used to detect when a rebuild is required.

• RealVector referenceCUBnds
  stores a reference copy of active continuous upper bounds when the approximation is built; used to detect when a rebuild is required.

• IntVector referenceDILBnds
  stores a reference copy of active discrete int lower bounds when the approximation is built; used to detect when a rebuild is required.

• IntVector referenceDIUBnds
stores a reference copy of active discrete int upper bounds when the approximation is built; used to detect when a rebuild is required.

- **RealVector referenceDRLBnds**
  stores a reference copy of active discrete real lower bounds when the approximation is built; used to detect when a rebuild is required.

- **RealVector referenceDRUBnds**
  stores a reference copy of active discrete real upper bounds when the approximation is built; used to detect when a rebuild is required.

- **RealVector referenceICVars**
  stores a reference copy of the inactive continuous variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- **IntVector referenceIDIVars**
  stores a reference copy of the inactive discrete int variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- **StringMultiArray referenceIDSVars**
  stores a reference copy of the inactive discrete string variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- **RealVector referenceIDRVars**
  stores a reference copy of the inactive discrete real variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

**Private Attributes**

- **Variables truthModelVars**
  copy of the truth model variables object used to simplify conversion among differing variable views in force_rebuild()

- **Constraints truthModelCons**
  copy of the truth model constraints object used to simplify conversion among differing variable views in force_rebuild()

- **bool mappingInitialized**
  track use of initialize_mapping() and finalize_mapping() due to potential redundancy between IteratorScheduler::run_iterator() and {Analyzer, Minimizer}::initialize_run()

**Additional Inherited Members**

13.206.1 Detailed Description

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).

The **SurrogateModel** class provides common functions to derived classes for computing and applying corrections to approximations.

13.206.2 Member Function Documentation

**int derived_evaluation_id ( ) const [inline], [protected], [virtual]**

return the current evaluation id for this Model.

return the SurrogateModel evaluation id counter. Due to possibly intermittent use of lower level components, this is not the same as approxInterface, actualModel, or orderedModels evaluation counts, which requires a consistent evaluation rekeying process.
13.206. SURROGATEMODEL CLASS REFERENCE

Reimplemented from Model.
References SurrogateModel::surrModelEvalCntr.

bool force_rebuild( ) [protected], [virtual]
evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data
This function forces a rebuild of the approximation according to the sub-model variables view, the approximation type, and whether the active approximation bounds or inactive variable values have changed since the last approximation build.
Reimplemented from Model.
References Constraints::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Variables::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), Variables::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_upper_bounds(), Variables::all_discrete_real_variables(), Variables::all_discrete_string_variables(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variables(), Constraints::copy(), Variables::copy(), Model::current_variables(), Model::currentVariables, Constraints::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Variables::discrete_int_variables(), Constraints::discrete_real_lower_bounds(), Constraints::discrete_real_upper_bounds(), Variables::discrete_real_variables(), Variables::discrete_string_variables(), Variables::inactive_continuous_variables(), Variables::inactive_discrete_int_variables(), Variables::inactive_discrete_real_variables(), Variables::inactive_discrete_string_variables(), Constraints::is_null(), Variables::is_null(), Model::is_null(), Model::model_type(), SurrogateModel::referenceCLBnds, SurrogateModel::referenceCUBnds, SurrogateModel::referenceDILBnds, SurrogateModel::referenceDIUBnds, SurrogateModel::referenceDRLBnds, SurrogateModel::referenceDRUBnds, SurrogateModel::referenceDISVars, Dakota::strbegins(), Model::subordinate_model(), Model::user_defined_constraints(), Model::userDefinedConstraints, and Variables::view().
Referenced by HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), and DataFitSurrModel::derived_evaluate_nowait().

13.206.3 Member Data Documentation

short responseMode [protected]
an enumeration that controls the response calculation mode in {DataFit,Hierarch}SurrModel approximate response computations
SurrBasedLocalMinimizer toggles this mode since compute_correction() does not back out old corrections.
Referenced by HierarchSurrModel::component_parallel_mode(), HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), DataFitSurrModel::derived_evaluate_nowait(), HierarchSurrModel::derived_set_communicators(), DataFitSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize_approx(), HierarchSurrModel::derived_synchronize_combine(), HierarchSurrModel::derived_synchronize_combine_nowait(), DataFitSurrModel::derived_synchronize_nowait(), HierarchSurrModel::derived_synchronize_nowait(), HierarchSurrModel::resize_response(), HierarchSurrModel::serve_run(), SurrogateModel::surrogate_response_mode(), HierarchSurrModel::surrogate_response_mode(), DataFitSurrModel::surrogate_response_mode(), SurrogateModel::SurrogateModel(), and HierarchSurrModel::update_from_subordinate_model().

size_t approxBuilds [protected]
number of calls to build_approximation()
used as a flag to automatically build the approximation if one of the derived evaluate functions is called prior to build_approximation().

Referenced by DataFitSurrModel::append_approximation(), DataFitSurrModel::approximation_coefficients(), HierarchSurrModel::build_approximation(), DataFitSurrModel::build_approximation(), HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), DataFitSurrModel::derived_evaluate_nowait(), HierarchSurrModel::init_model(), DataFitSurrModel::init_model(), DataFitSurrModel::pop_approximation(), DataFitSurrModel::update_approximation(), and DataFitSurrModel::update_from_model().

The documentation for this class was generated from the following files:

- SurrogateModel.hpp
- SurrogateModel.cpp

### 13.207 SysCallApplicInterface Class Reference

Derived application interface class which spawns simulation codes using system calls.

Inheritance diagram for SysCallApplicInterface:

```
+---------------------------+ +---------------------------+ +---------------------------+
| Interface                 | | ApplicationInterface     | | ProcessApplicInterface   |
|                           | +---------------------------+ | SysCallApplicInterface   |
|                           |                             | GridApplicInterface      |
```

**Public Member Functions**

- `SysCallApplicInterface (const ProblemDescDB &problem_db)`
  
  *constructor*

- `~SysCallApplicInterface ()`
  
  *destructor*

**Protected Member Functions**

- `void wait_local_evaluations (PRPQueue &prp_queue)`
- `void test_local_evaluations (PRPQueue &prp_queue)`
- `int synchronous_local_analysis (int analysis_id)`
- `void init_communicators_checks (int max_eval_concurrency)`
- `void set_communicators_checks (int max_eval_concurrency)`
- `void map_bookkeeping (pid_t pid, int fn_eval_id)`

  *bookkeeping of process and evaluation ids for asynchronous maps*
• pid_t create_evaluation_process (bool block_flag)
  
  Spawn the evaluation by managing the input filter, analysis drivers, and output filter. Called from derived_map() & derived_map_asynch().

Private Member Functions

• bool system_call_file_test (const bfs::path &root_file)
  
  detect completion of a function evaluation through existence of the necessary results file(s)

• void spawn_evaluation_to_shell (bool block_flag)
  
  spawn a complete function evaluation

• void spawn_input_filter_to_shell (bool block_flag)
  
  spawn the input filter portion of a function evaluation

• void spawn_analysis_to_shell (int analysis_id, bool block_flag)
  
  spawn a single analysis as part of a function evaluation

• void spawn_output_filter_to_shell (bool block_flag)
  
  spawn the output filter portion of a function evaluation

Private Attributes

• IntSet sysCallSet
  
  set of function evaluation id's for active asynchronous system call evaluations

• IntShortMap failCountMap
  
  map linking function evaluation id's to number of response read failures

Additional Inherited Members

13.207.1 Detailed Description

Derived application interface class which spawns simulation codes using system calls. system() is part of the C API and can be used on both Windows and Unix systems.

13.207.2 Member Function Documentation

void wait_local_evaluations ( PRPQueue & prp_queue ) [inline], [protected], [virtual]

Check for completion of active asynch jobs (tracked with sysCallSet). Wait for at least one completion and complete all jobs that have returned. This satisfies a "fairness" principle, in the sense that a completed job will always be processed (whereas accepting only a single completion could always accept the same completion - the case of very inexpensive fn. evals. - and starve some servers).

Reimplemented from ApplicationInterface.

References ApplicationInterface::completionSet, and SysCallApplicInterface::test_local_evaluations().
void test_local_evaluations ( PRPQueue & prp_queue ) [protected], [virtual]
Check for completion of active async jobs (tracked with sysCallSet). Make one pass through sysCallSet &
complete all jobs that have returned.
Reimplemented from ApplicationInterface.
References Dakota::abort_handler(), Response::active_set(), ApplicationInterface::completionSet, SysCallApplic-
Interface::failCountMap, ProcessApplicInterface::fileNameMap, Interface::final_eval_id_tag(), Dakota::lookup_-
by_eval_id(), ApplicationInterface::manage_failure(), ProcessApplicInterface::read_results_files(), SysCallApplic-
Interface::sysCallSet, and SysCallApplicInterface::system_call_file_test().
Referenced by SysCallApplicInterface::wait_local_evaluations().

int synchronous_local_analysis ( int analysis_id ) [inline], [protected], [virtual]
This code provides the derived function used by ApplicationInterface::serve_analyses_synch().
Reimplemented from ApplicationInterface.
References SysCallApplicInterface::spawn_analysis_to_shell().

void init_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
No derived interface plug-ins, so perform construct-time checks. However, process init issues as warnings since
some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used.
Reimplemented from ApplicationInterface.
References ApplicationInterface::check_multiprocessor_analysis().

void set_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
Process run-time issues as hard errors.
Reimplemented from ApplicationInterface.
References Dakota::abort_handler(), and ApplicationInterface::check_multiprocessor_analysis().

void spawn_evaluation_to_shell ( bool block_flag ) [private]
spawn a complete function evaluation
Put the SysCallApplicInterface to the shell. This function is used when all portions of the function evaluation
(i.e., all analysis drivers) are executed on the local processor.
References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, ProcessApplicInterface-::curWorkdir, Dakota::flush(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::multipleParamsFiles,
ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare-
_process_environment(), ProcessApplicInterface::programNames, ProcessApplicInterface::reset_process_environment(),
ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppress-
Output, and ProcessApplicInterface::useWorkdir.
Referenced by SysCallApplicInterface::create_evaluation_process().

void spawn_input_filter_to_shell ( bool block_flag ) [private]
spawn the input filter portion of a function evaluation
Put the input filter to the shell. This function is used when multiple analysis drivers are spread between
processors. No need to check for a Null input filter, as this is checked externally. Use of nonblocking shells is
supported in this fn, although its use is currently prevented externally.
References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, Dakota::flush(), ProcessApplicInterface::FilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare_process_environment(), ProcessApplicInterface::reset_process_environment(), ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), and ApplicationInterface::suppressOutput.

Referenced by SysCallApplicInterface::create_evaluation_process().

```cpp
void spawn_analysis_to_shell ( int analysis_id, bool block_flag ) [private]
```

spawn a single analysis as part of a function evaluation

Put a single analysis to the shell. This function is used when multiple analysis drivers are spread between processors. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.

References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, Dakota::flush(), ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare_process_environment(), ProcessApplicInterface::programNames, ProcessApplicInterface::reset_process_environment(), ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), and ApplicationInterface::suppressOutput.

Referenced by SysCallApplicInterface::create_evaluation_process(), SysCallApplicInterface::synchronous_local_analysis(), and GridApplicInterface::synchronous_local_analysis().

```cpp
void spawn_output_filter_to_shell ( bool block_flag ) [private]
```

spawn the output filter portion of a function evaluation

Put the output filter to the shell. This function is used when multiple analysis drivers are spread between processors. No need to check for a Null output filter, as this is checked externally. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.

References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, Dakota::flush(), ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare_process_environment(), ProcessApplicInterface::reset_process_environment(), ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), and ApplicationInterface::suppressOutput.

Referenced by SysCallApplicInterface::create_evaluation_process().

The documentation for this class was generated from the following files:

- SysCallApplicInterface.hpp
- SysCallApplicInterface.cpp

### 13.208 TabularDataTruncated Class Reference

exception thrown when data read truncated

Inheritance diagram for TabularDataTruncated:

```
  FileReadException
    TabularDataTruncated
```

#### Public Member Functions

- **TabularDataTruncated** (const std::string &msg)
13.208.1 Detailed Description

exception thrown when data read truncated

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp

13.209 TabularReader Class Reference

Utility used in derived read_core to read values in tabular format.

Public Member Functions

- template<typename ArrayType>
  void operator()(std::istream &s, size_t start_index, size_t num_items, ArrayType &array_data, StringMultiArrayView label_array)

13.209.1 Detailed Description

Utility used in derived read_core to read values in tabular format.

13.209.2 Member Function Documentation

void operator()(std::istream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array) [inline]

The tabular reader doesn’t forward the label arrays

The documentation for this class was generated from the following file:

- DakotaVariables.hpp

13.210 TabularWriter Class Reference

Utility used in derived write_core to write values in tabular format.

Public Member Functions

- template<typename ArrayType>
  void operator()(std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)

13.210.1 Detailed Description

Utility used in derived write_core to write values in tabular format.
13.210.2 Member Function Documentation

```cpp
void operator() ( std::ostream & s, size_t start_index, size_t num_items, const ArrayType & array_data, StringMultiArrayConstView label_array ) [inline]
```

The tabular writer doesn’t forward the label arrays.

The documentation for this class was generated from the following file:

- DakotaVariables.hpp

13.211 TANA3Approximation Class Reference

Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation).

Inheritance diagram for TANA3Approximation:

```
Approximation

\[ \uparrow \]

TANA3Approximation
```

Public Member Functions

- **TANA3Approximation ()**
  - default constructor
- **TANA3Approximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  - standard constructor
- **TANA3Approximation (const SharedApproxData &shared_data)**
  - alternate constructor
- **~TANA3Approximation ()**
  - destructor

Protected Member Functions

- **int min_coefficients () const**
  - return the minimum number of samples (unknowns) required to build the derived class approximation type in num- Vars dimensions
- **int num_constraints () const**
  - return the number of constraints to be enforced via an anchor point
- **void build (size_t index=NPOS)**
  - builds the approximation from scratch
- **Real value (const Variables &vars)**
  - retrieve the approximate function value for a given parameter vector
- **const RealVector & gradient (const Variables &vars)**
  - retrieve the approximate function gradient for a given parameter vector
- **void clear_current ()**
CHAPTER 13. CLASS DOCUMENTATION

Private Member Functions

- void find_scaled_coefficients ()
  compute TANA coefficients based on scaled inputs
- void offset (const RealVector &x, RealVector &s)
  based on minX, apply offset scaling to x to define s

Private Attributes

- RealVector pExp
  vector of exponent values
- RealVector minX
  vector of minimum parameter values used in scaling
- RealVector scX1
  vector of scaled x1 values
- RealVector scX2
  vector of scaled x2 values
- Real H
  the scalar Hessian value in the TANA-3 approximation

Additional Inherited Members

13.211.1 Detailed Description

Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation). The TANA3Approximation class provides a multipoint approximation based on matching value and gradient data from two points (typically the current and previous iterates) in parameter space. It forms an exponential approximation in terms of intervening variables.

13.211.2 Member Function Documentation

void build ( size_t index = _NPOS ) [protected], [virtual]
builds the approximation from scratch
This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.
Reimplemented from Approximation.
References Dakota::abort_handler(), Approximation::approxData, Approximation::build(), TANA3Approximation::find_scaled_coefficients(), TANA3Approximation::minX, SharedApproxData::numVars, TANA3Approximation::pExp, and Approximation::sharedDataRep.

void clear_current ( ) [inline], [protected], [virtual]
Redefine default implementation to support history mechanism.
Reimplemented from Approximation.
References Approximation::approxData.
The documentation for this class was generated from the following files:
- TANA3Approximation.hpp
- TANA3Approximation.cpp
13.212  TaylorApproximation Class Reference

Derived approximation class for first- or second-order Taylor series (a local approximation).

Inheritance diagram for TaylorApproximation:

```
<table>
<thead>
<tr>
<th>Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaylorApproximation</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **TaylorApproximation ()**
  default constructor
- **TaylorApproximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  standard constructor
- **TaylorApproximation (const SharedApproxData &shared_data)**
  alternate constructor
- **~TaylorApproximation ()**
  destructor

Protected Member Functions

- **int min_coefficients () const**
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num- Vars dimensions
- **void build (size_t index=NPOS)**
  builds the approximation from scratch
- **Real value (const Variables &vars)**
  retrieve the approximate function value for a given parameter vector
- **const RealVector & gradient (const Variables &vars)**
  retrieve the approximate function gradient for a given parameter vector
- **const RealSymMatrix & hessian (const Variables &vars)**
  retrieve the approximate function Hessian for a given parameter vector

Additional Inherited Members

13.212.1  Detailed Description

Derived approximation class for first- or second-order Taylor series (a local approximation).

The TaylorApproximation class provides a local approximation based on data from a single point in parameter space. It uses a zeroth-, first- or second-order Taylor series expansion: \( f(x) = f(x_c) \) for zeroth-order, plus \( \text{grad}(x-x_c) \) for first- and second-order, and plus \( (x-x_c)^T \) \( \text{Hess}(x_c) (x-x_c) / 2 \) for second-order.
13.212.2 Member Function Documentation

```c
void build ( size_t index = _NPOS ) [protected], [virtual]
```
builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from `Approximation`.

References Dakota::abort_handler(), Approximation::approxData, Approximation::build(), SharedApproxData::buildDataOrder, SharedApproxData::numVars, and Approximation::sharedDataRep.

The documentation for this class was generated from the following files:

- TaylorApproximation.hpp
- TaylorApproximation.cpp

13.213 TestDriverInterface Class Reference

Inheritance diagram for TestDriverInterface:

```
  Interface
   ├── ApplicationInterface
   │    ├── DirectApplicInterface
   │    └── TestDriverInterface
```

**Public Member Functions**

- `TestDriverInterface (const ProblemDescDB &problem_db)`
  constructor
- `~TestDriverInterface ()`
  destructor

**Protected Member Functions**

- virtual `int derived_map_ac (const Dakota::String &ac_name)`
  execute an analysis code portion of a direct evaluation invocation

**Private Member Functions**

- `int cantilever ()`
  scaled cantilever test function for optimization
- `int mod_cantilever ()`
  unscaled cantilever test function for UQ
13.213. TESTDRIVERINTERFACE CLASS REFERENCE

• int cyl_head ()
  the cylinder head constrained optimization test fn
• int multimodal ()
  multimodal UQ test function
• int log_ratio ()
  the log_ratio UQ test function
• int short_column ()
  the short_column UQ/OUU test function
• int lf_short_column ()
  a low fidelity short_column test function
• int mf_short_column ()
  alternate short_column formulations for multifidelity or model form studies
• int alternate_short_column_forms (int form)
  helper fn for alternate forms
• int side_impact_cost ()
  the side_impact_cost UQ/OUU test function
• int side_impact_perf ()
  the side_impact_perf UQ/OUU test function
• int rosenbrock ()
  the Rosenbrock optimization and least squares test fn
• int modified_rosenbrock ()
  the modified Rosenbrock optimization and least squares test fn. The modification is the addition of an sin^2 term so that function can not be exactly approximated by a low degree polynomial
• int generalized_rosenbrock ()
  n-dimensional Rosenbrock (Schittkowski)
• int extended_rosenbrock ()
  n-dimensional Rosenbrock (Nocedal/Wright)
• int lf_rosenbrock ()
  a low fidelity version of the Rosenbrock function
• int extra_lf_rosenbrock ()
  an extra low fidelity version of the Rosenbrock function
• int mf_rosenbrock ()
  alternate Rosenbrock formulations for multifidelity or model form studies
• int lf_poly_prod ()
  modified low fidelity Rosenbrock to test SBO with hierarchical approximations
• int poly_prod ()
  modified low fidelity Rosenbrock to test SBO with hierarchical approximations
• int gerstner ()
  the isotropic/anisotropic Gerstner test function family
• int scalable_gerstner ()
  scalable versions of the Gerstner test family
• void get_genz_coefficients (int num_dims, Real factor, int c_type, RealVector &c, RealVector &w)
define coefficients needs for genz model

- `int genz()`
  scalable test functions from the Genz test suite
- `int damped_oscillator()`
  1d-6d that returns field values (ode solution)
- `int steady_state_diffusion_1d()`
  solve the 1d steady-state diffusion eqn < with uncertain field diffusivity
- `int transient_diffusion_1d()`
  solve the 1d transient diffusion equation < with uncertain scalar diffusivity
- `int predator_prey()`
- `int steel_column_cost()`
  solve a predator prey population dynamics model
- `int steel_column_perf()`
  the steel_column_perf UQ/OUU test function
- `int sobol_rational()`
  Sobol SA rational test function.
- `int sobol_g_function()`
  Sobol SA discontinuous test function.
- `int sobol_ishigami()`
  Sobol SA transcendental test function.
- `int text_book()`
  the text_book constrained optimization test function
- `int text_book1()`
  portion of text_book() evaluating the objective fn
- `int text_book2()`
  portion of text_book() evaluating constraint 1
- `int text_book3()`
  portion of text_book() evaluating constraint 2
- `int text_book_ouu()`
  the text_book_ouu OUU test function
- `int scalable_text_book()`
  scalable version of the text_book test function
- `int scalable_monomials()`
  simple monomials for UQ exactness testing
- `int mogatest1()`
  MOP2 from Van Veldhuizen, pp. 5-13.
- `int mogatest2()`
- `int mogatest3()`
  Srinivas’ from Van Veldhuizen, pp. B-5.
- `int illumination()`
  illumination example in Boyd as a general < minimization problem
- `int barnes()`
barnes test for SBO performance from Rodriguez, < Perez, Renaud, et al.

- int barnes_lf ()
  
  lo-fi barnes test for SBO performance

- void herbie1D (size_t der_mode, Real xc_loc, std::vector<Real> &w_and_ders)
  
  1D components of herbie function

- void smooth_herbie1D (size_t der_mode, Real xc_loc, std::vector<Real> &w_and_ders)
  
  1D components of smooth_herbie function

- void shubert1D (size_t der_mode, Real xc_loc, std::vector<Real> &w_and_ders)
  
  1D components of shubert function

- int herbie ()
  
  returns the N-D herbie function

- int smooth_herbie ()
  
  returns the N-D smooth herbie function

- int shubert ()
  
  returns the N-D shubert function

- int bayes_linear ()
  
  Scalable test function for Bayesian methods, to estimate parameters.

- void separable_combine (Real mult_scale_factor, std::vector<Real> &w, std::vector<Real> &d1w, std::vector<Real> &d2w)
  
  utility to combine components of separable fns

- Real levenshtein_distance (const String &v)
  
  Compute Levenshtein distance between v and LEV_REF.

- int salinas ()
  
  direct interface to the SALINAS structural dynamics code

- int mc_api_run ()
  
  direct interface to ModelCenter via API, HKIM 4/3/03

- int aniso_quad_form ()
  
  1-D function using an anisotropic quadratic < form

Static Private Attributes

- static StringRealMap levenshteinDistanceCache
  
  Cache results of Levenshtein distance calc for efficiency.

Additional Inherited Members

13.213.1 Detailed Description

Specialization of DirectApplicInterface to embed algebraic test function drivers directly in Dakota
13.2.13.2 Member Function Documentation

```cpp
int derived_map_ac ( const Dakota::String & ac_name ) [protected], [virtual]
```
execute an analysis code portion of a direct evaluation invocation
   Derived map to evaluate a particular built-in test analysis function
   Reimplemented from DirectApplicInterface.
   References Dakota::abort_handler(), ApplicationInterface::analysisServerId, TestDriverInterface::aniso_quad_form(), TestDriverInterface::barnes(), TestDriverInterface::barnes_lf(), TestDriverInterface::bayes_linear(), TestDriverInterface::cantilever(), TestDriverInterface::cyl_head(), TestDriverInterface::damped_oscillator(), DirectApplicInterface::driverTypeMap, TestDriverInterface::extended_rosenbrock(), TestDriverInterface::extra_lf_rosenbrock(), TestDriverInterface::generalized_rosenbrock(), TestDriverInterface::genz(), TestDriverInterface::gerstner(), TestDriverInterface::herbie(), TestDriverInterface::illumination(), TestDriverInterface::lf_poly_prod(), TestDriverInterface::lf_rosenbrock(), TestDriverInterface::lf_short_column(), TestDriverInterface::log_ratio(), TestDriverInterface::mc_api_run(), TestDriverInterface::mf_rosenbrock(), TestDriverInterface::mf_short_column(), TestDriverInterface::modified_cantilever(), TestDriverInterface::modified_rosenbrock(), TestDriverInterface::mogatest1(), TestDriverInterface::mogatest2(), TestDriverInterface::mogatest3(), TestDriverInterface::poly_prod(), TestDriverInterface::poly_roux(), TestDriverInterface::poly_rouxl(), TestDriverInterface::poly_rouxll(), TestDriverInterface::poly_rouxlll(), TestDriverInterface::poly_rouxllll(), TestDriverInterface::poly_quad(), TestDriverInterface::scaled(), TestDriverInterface::scaled_text_book(), TestDriverInterface::scaled_text_book1(), TestDriverInterface::scaled_text_book2(), TestDriverInterface::scaled_text_book3(), TestDriverInterface::scaled_text_book_ouu(), TestDriverInterface::scaled_text_book_ouu().

```cpp
int lf_poly_prod ( ) [private]
```
modified low fidelity Rosenbrock to test SBO with < hierarchical approximations
   modified lo-fi Rosenbrock to test SBO with hierarchical approximations
   References Dakota::abort_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numFns, and DirectApplicInterface::xC.
   Referenced by TestDriverInterface::derived_map_ac().

```cpp
int poly_prod ( ) [private]
```
modified low fidelity Rosenbrock to test SBO with < hierarchical approximations
   modified lo-fi Rosenbrock to test SBO with hierarchical approximations
   References Dakota::abort_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numFns, and DirectApplicInterface::xC.
   Referenced by TestDriverInterface::derived_map_ac().

```cpp
int steady_state_diffusion_1d ( ) [private]
```
solve the 1d steady-state diffusion eqn < with uncertain field diffusivity
   Solve the 1D diffusion equation with an uncertain variable coefficient using the spectral Chebyshev collocation method.
\[ \text{del}(k \text{ del}(u) ) = f \text{ on } [0,1] \text{ subject to } u(0) = 0 \text{ } u(1) = 0 \]

Here we set \( f = -1 \) and \( k = 1 + 4 + \sum_{d} \left[ \cos(2\pi x)/(\pi d)^2 + z[d] \right] d=1,...,\text{num\_dims} \) where \( z_d \) are random variables, typically i.i.d uniform[-1,1].

References Dakota::NPOS, Dakota::abort\_handler(), Dakota::find\_index(), DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numADIV, DirectApplicInterface::numFns, DirectApplicInterface::numVars, DirectApplicInterface::xC, DirectApplicInterface::xDI, DirectApplicInterface::xDILabels, DirectApplicInterface::xDR, DirectApplicInterface::xDLabels, DirectApplicInterface::xDLabels, DirectApplicInterface::xDS, and DirectApplicInterface::xDSLLabels.

Referenced by TestDriverInterface::derived\_map\_ac().

```cpp
int steel\_column\_cost() [private]
```

Solve a predator prey population dynamics model

the steel\_column\_cost UQ/OUU test function

References Dakota::abort\_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, DirectApplicInterface::numVars, DirectApplicInterface::varTypeDVV, and DirectApplicInterface::xC.

Referenced by TestDriverInterface::derived\_map\_ac().

```cpp
int barnes() [private]
```

Barnes test for SBO performance from Rodriguez, Perez, Renaud, et al.

barnes test for SBO performance from Rodriguez, Perez, Renaud, et al.

References Dakota::abort\_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::directFnD\_VV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, and DirectApplicInterface::xC.

Referenced by TestDriverInterface::derived\_map\_ac().

```cpp
int barnes\_lf() [private]
```

Lo-fi Barnes test for SBO performance

Lo-fi Barnes test for SBO performance from Rodriguez, Perez, Renaud, et al.

References Dakota::abort\_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::directFnD\_VV, DirectApplicInterface::fnGrads, DirectApplicInterface::gradFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, and DirectApplicInterface::xC.

Referenced by TestDriverInterface::derived\_map\_ac().

```cpp
void herbie\_1D( size\_t der\_mode, Real xc\_loc, std::vector< Real > & w\_and\_ders ) [private]
```

1D components of herbie function

1D Herbie function and its derivatives (apart from a multiplicative factor)

Referenced by TestDriverInterface::herbie().
void smooth_herbies1D ( size_t der_mode, Real xc_loc, std::vector<Real> & w_and_ders ) [private]

1D components of smooth_herbies function
1D Smoothed Herbie= 1DHerbie minus the high frequency sine term, and its derivatives (apart from a multiplicative factor)
Referenced by TestDriverInterface::smooth_herbies().

void shubert1D ( size_t der_mode, Real xc_loc, std::vector<Real> & w_and_ders ) [private]

1D components of shubert function
1D Shubert function and its derivatives (apart from a multiplicative factor)
Referenced by TestDriverInterface::shubert().

int herbie ( ) [private]
returns the N-D herbie function
N-D Herbie function and its derivatives.
References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, TestDriverInterface::herbie1D(), DirectApplicInterface::numDerivVars, DirectApplicInterface::numVars, TestDriverInterface::separable_combine(), and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

int smooth_herbies ( ) [private]
returns the N-D smooth herbies function
N-D Smoothed Herbies function and its derivatives.
References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, DirectApplicInterface::numDerivVars, DirectApplicInterface::numVars, TestDriverInterface::separable_combine(), TestDriverInterface::smooth_herbies1D(), and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

void separable_combine ( Real mult_scale_factor, std::vector<Real> & w, std::vector<Real> & d1w, std::vector<Real> & d2w ) [private]
utility to combine components of separable fns
this function combines N 1D functions and their derivatives to compute a N-D separable function and its derivatives, logic is general enough to support different 1D functions in different dimensions (can mix and match)
References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::numDerivVars, and DirectApplicInterface::numVars.
Referenced by TestDriverInterface::herbie(), TestDriverInterface::shubert(), and TestDriverInterface::smooth_herbies().

Real levenshtein_distance ( const String & v ) [private]
Compute Levenshtein distance between v and LEV_REF.
Levenshtein distance is the number of changes (single character
References Dakota::LEV_REF, and TestDriverInterface::levenshteinDistanceCache.
Referenced by TestDriverInterface::text_book1(), TestDriverInterface::text_book2(), and TestDriverInterface::text_book3().
int mc_api_run( ) [private]

direct interface to ModelCenter via API, HKIM 4/3/03

The ModelCenter interface doesn’t have any specific construct vs. run time functions. For now, we manage it
along with the integrated test drivers

References Dakota::abort_handler, DirectApplicInterface::analysisComponents, DirectApplicInterface::analysis-
DriverIndex, Dakota::dc_ptr_int, DirectApplicInterface::directFnASV, Interface::fnLabels, DirectApplicInterface-
::fnVals, Dakota::mc_ptr_int, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, Direct-
ApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numFns, DirectApplicInterface-
::xC, DirectApplicInterface::xCLabels, DirectApplicInterface::xDI, DirectApplicInterface::xDILabels, DirectApplic-
Interface::xDR, and DirectApplicInterface::xDRLabels.

Referenced by TestDriverInterface::derived_map_ac().

The documentation for this class was generated from the following files:

- TestDriverInterface.hpp
- TestDriverInterface.cpp

13.214 TrackerHTTP Class Reference

TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.

Classes

- struct Server
  
  struct to hold tracker/proxy pairs

Public Member Functions

- TrackerHTTP()
  
  default constructor is allowed, but doesn’t generate output

- TrackerHTTP(int world_rank=0)
  
  standard constructor with ProblemDescDB, rank

- TrackerHTTP()

  destructor to free handles

- void post_start(ProblemDescDB &problem_db)

  post the start of an analysis and archive start time

- void post_finish(unsigned runtime=0)

  post the completion of an analysis including elapsed time

Private Member Functions

- void initialize(int world_rank=0)

  shared initialization functions across constructors

- void url_add_field(std::string &url, const char *keyword, const std::string &value, bool delimit=true) const

  append keyword/value pair to url in GET style (with &keyword=value); set delimit = false to omit the &

- void build_default_data(std::string &url, std::string &mode, const std::string &time_t, const std::string &rawtime, const std::string &time_t) const
construct URL with shared information for start/finish

- void send_data_using_get (const std::string &urltopost) const
  transmit data to the web server using GET

- void send_data_using_post (const std::string &datatopost)
  POST separate location and query; datatopost="name=daniel&project=curl"

- void split_string (const std::string &s, const char &delim, std::vector<std::string> &elems)
  Split a string on a delimiter and place tokens in elems.

- void parse_tracking_string (const std::string &dt)
  Populate serverList with tracker and proxy URLs from dt.

- void populate_method_list (ProblemDescDB &problem_db)
  extract list of methods from problem database

- std::string get_uid () const
  get the real user ID

- std::string get_username () const
  get the username as reported by the environment

- std::string get_hostname () const
  get the system hostname

- std::string get_os () const
  get the operating system

- std::string get_datetime (const std::time_t &rawtime) const
  get the date and time as a string YYYYMMDDHHMMSS

**Private Attributes**

- CURL * curlPtr
  pointer to the curl handler instance

- FILE * devNull
  pointer to /dev/null

- std::list< Server > serverList
  List of servers to try (tracker and proxy)

- long timeoutSeconds
  seconds until the request will timeout (may have issues with signals)

- std::string methodList
  list of active methods

- std::string dakotaVersion
  DAKOTA version.

- std::time_t startTime
  cached starting time in raw seconds

- short outputLevel
  verbosity control

**13.214.1 Detailed Description**

TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.
13.214.2 Member Function Documentation

```cpp
void send_data_using_get ( const std::string & urltopost ) const  
  [private]
```

transmit data to the web server using GET
whole url including location & fields
References TrackerHTTP::curlPtr, and TrackerHTTP::outputLevel.

```cpp
void send_data_using_post ( const std::string & datatopost )  
  [private]
```

POST separate location and query, datatopost="name=daniel&project=curl".

References TrackerHTTP::curlPtr, TrackerHTTP::outputLevel, and TrackerHTTP::serverList.

Referenced by TrackerHTTP::post_finish(), and TrackerHTTP::post_start().

The documentation for this class was generated from the following files:

- TrackerHTTP.hpp
- TrackerHTTP.cpp

13.215 TraitsBase Class Reference

Base class for traits.

Inheritance diagram for TraitsBase:
Public Member Functions

- **TraitsBase ()**
  
  *default constructor*

- virtual `~TraitsBase ()`
  
  *destructor*

- virtual bool `is_derived ()`
  
  *A temporary query used in the refactor.*

- virtual bool `requires_bounds ()`
  
  *Return the flag indicating whether method requires bounds.*

- virtual bool `supports_linear_equality ()`
  
  *Return the flag indicating whether method supports linear equalities.*

- virtual bool `supports_linear_inequality ()`
  
  *Return the flag indicating whether method supports linear inequalities.*

- virtual LIN**EAR**_INEQUALITY_FORMAT `linear_inequality_format ()`
  
  *Return the format used for linear inequality constraints.*

- virtual bool `supports_nonlinear_equality ()`
Return the flag indicating whether method supports nonlinear equalities.

- virtual NONLINEAR_EQUALITY_FORMAT nonlinear_equality_format ()
  Return the format used for nonlinear equality constraints.

- virtual bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.

- virtual NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  Return the format used for nonlinear inequality constraints.

- virtual bool expects_nonlinear_inequalities_first ()
  Return the flag indicating whether method expects nonlinear inequality constraints followed by nonlinear equality constraints.

- virtual bool supports_scaling ()
  Return the flag indicating whether method supports parameter scaling.

- virtual bool supports_least_squares ()
  Return the flag indicating whether method supports least squares.

- virtual bool supports_multiobjectives ()
  Return flag indicating whether method supports multiobjective optimization.

- virtual bool supports_continuous_variables ()
  Return the flag indicating whether method supports continuous variables.

- virtual bool supports_discrete_variables ()
  Return the flag indicating whether method supports continuous variables.

- virtual bool provides_best_objective ()
  Return the flag indicating whether method provides best objective result.

- virtual bool provides_best_parameters ()
  Return the flag indicating whether method provides best parameters result.

- virtual bool provides_best_constraint ()
  Return the flag indicating whether method provides best constraint result.

- virtual bool provides_final_gradient ()
  Return the flag indicating whether method provides final gradient result.

- virtual bool provides_final_hessian ()
  Return the flag indicating whether method provides final hessian result.

13.215.1 Detailed Description

Base class for traits.

TraitsBase provides default traits through various accessors.

The documentation for this class was generated from the following files:

- DakotaTraitsBase.hpp
- DakotaTraitsBase.cpp

13.216 UsageTracker Class Reference

Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL.
Public Member Functions

- **UsageTracker ()**
  
  *default construction: no output*

- **UsageTracker (int world_rank)**
  
  *standard constructor; will output on rank 0*

- **void post_start (ProblemDescDB &problem_db)**
  
  *post the start of an analysis and archive start time*

- **void post_finish (unsigned runtime=0)**
  
  *post the completion of an analysis including elapsed time*

Private Member Functions

- **UsageTracker (const UsageTracker &)**
  
  *copy construction is disallowed*

Private Attributes

- **boost::shared_ptr< TrackerHTTP > pTrackerHTTP**
  
  *posts usage data to Web server; using shared_ptr due to potentially incomplete type and requirements for checked-delete in debug builds (scoped_ptr would suffice)*

13.216.1 Detailed Description

Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL.

All conditional compilation is managed in the cpp file; all operations are no-op in this wrapper if not enabling tracking...

13.216.2 Constructor & Destructor Documentation

**UsageTracker ( int world_rank )**

standard constructor; will output on rank 0

  *standard constructor; will output on rank 0 and only initializes if tracking compiled in and not disable by environment*

  *References UsageTracker::pTrackerHTTP.*

  *The documentation for this class was generated from the following files:*

  - UsageTracker.hpp
  - UsageTracker.cpp

13.217 Var_ichck Struct Reference

structure for verifying bounds and initial point for string-valued vars
13.218. VAR_RCHECK STRUCT REFERENCE

Public Attributes

- const char ∗ name
- size_t DataVariablesRep::* n
- void(* vgen)(DataVariablesRep *, size_t)
- IntVector DataVariablesRep::* L
- IntVector DataVariablesRep::* U
- IntVector DataVariablesRep::* V
- StringArray DataVariablesRep::* Lbl

13.217.1 Detailed Description

structure for verifying bounds and initial point for string-valued vars
structure for verifying bounds and initial point for integer-valued vars
The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

13.218 Var_rcheck Struct Reference

structure for verifying bounds and initial point for real-valued vars

Public Attributes

- const char ∗ name
- size_t DataVariablesRep::* n
- void(* vgen)(DataVariablesRep *, size_t)
- RealVector DataVariablesRep::* L
- RealVector DataVariablesRep::* U
- RealVector DataVariablesRep::* V
- StringArray DataVariablesRep::* Lbl

13.218.1 Detailed Description

structure for verifying bounds and initial point for real-valued vars
The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

13.219 Variables Class Reference

Base class for the variables class hierarchy.
Inheritance diagram for Variables:
Public Member Functions

- `Variables ()`
  `default constructor`
- `Variables (const ProblemDescDB &problem_db)`
  `standard constructor (explicit disallows its use for implicit type conversion)`
- `Variables (const SharedVariablesData &svd)`
  `alternate constructor for instantiations on the fly (explicit disallows its use for implicit type conversion)`
- `Variables (const Variables &vars)`
  `copy constructor`
- `virtual ~Variables ()`
  `destructor`
- `Variables operator= (const Variables &vars)`
  `assignment operator`
- `virtual void read (std::istream &s)`
  `read a variables object from an std::istream`
- `virtual void write (std::ostream &s, unsigned short vars_part=ALL_VARS) const`
  `write a variables object to an std::ostream, e.g., the console, optionally specifying which partition (all/active/inactive)`
- `virtual void write_aprepro (std::ostream &s) const`
  `write a variables object to an std::ostream in aprepro format, e.g., a parameters file`
- `virtual void read.annotated (std::istream &s)`
  `read a variables object in annotated format from an istream`
- `virtual void write.annotated (std::ostream &s) const`
  `write a variables object in annotated format to an std::ostream`
- `virtual void read_tabular (std::istream &s, unsigned short vars_part=ALL_VARS)`
  `read a variables object in tabular format from an istream, optionally specifying which partition (all/active/inactive)`
- `virtual void write_tabular (std::ostream &s, unsigned short vars_part=ALL_VARS) const`
  `write a variables object in tabular format to an std::ostream, optionally specifying which partition (all/active/inactive)`
- `virtual void write_tabular_labels (std::ostream &s, unsigned short vars_part=ALL_VARS) const`
  `write the labels in input spec order to a std::ostream, optionally specifying which partition (all/active/inactive)`
- `virtual void read (MPIUnpackBuffer &s)`
  `read a variables object from a packed MPI buffer`
- `virtual void write (MPIPackBuffer &s) const`
  `write a variables object to a packed MPI buffer`
- `size_t tv () const`
  `total number of vars`
- `size_t cv () const`
  `number of active continuous vars`
- `size_t cv_start () const`
  `start index of active continuous vars`
- `size_t div () const`
  `number of active discrete int vars`
- `size_t div_start () const`
start index of active discrete int vars
- size_t dsv () const
  number of active discrete string vars
- size_t dsv_start () const
  start index of active discrete string vars
- size_t drv () const
  number of active discrete real vars
- size_t drv_start () const
  start index of active discrete real vars
- size_t icv () const
  number of inactive continuous vars
- size_t icv_start () const
  start index of inactive continuous vars
- size_t idiv () const
  number of inactive discrete int vars
- size_t idiv_start () const
  start index of inactive discrete int vars
- size_t idsv () const
  number of inactive discrete string vars
- size_t idsv_start () const
  start index of inactive discrete string vars
- size_t idrv () const
  number of inactive discrete real vars
- size_t idrv_start () const
  start index of inactive discrete real vars
- size_t acv () const
  total number of continuous vars
- size_t adiv () const
  total number of discrete integer vars
- size_t adsv () const
  total number of discrete string vars
- size_t adrv () const
  total number of discrete real vars
- const SharedVariablesData & shared_data () const
  return sharedVarsData
- SharedVariablesData & shared_data ()
  return sharedVarsData
- void shape ()
  shape a Variables object based on sharedVarsData
- void reshape ()
  reshape an existing Variables object based on updated sharedVarsData
- Real continuous_variable (size_t index) const
  return an active continuous variable
• const RealVector & continuous_variables() const
  return the active continuous variables (Note: returns a view by const reference, but initializing a RealVector from this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable view, use continuous_variables_view())
• void continuous_variable(Real c_var, size_t index)
  set an active continuous variable
• void continuous_variables(const RealVector &c_vars)
  set the active continuous variables
• int discrete_int_variable(size_t index) const
  return an active discrete integer variable
• const IntVector & discrete_int_variables() const
  return the active discrete integer variables (Note: returns a view by const reference, but initializing an IntVector from this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable view, use discrete_int_variables_view())
• void discrete_int_variable(int di_var, size_t index)
  set an active discrete integer variable
• void discrete_int_variables(const IntVector &di_vars)
  set the active discrete integer variables
• const String & discrete_string_variable(size_t index) const
  return an active discrete string variable
• StringMultiArrayConstView discrete_string_variables() const
  return the active discrete string variables (Note: returns a view by const reference, but initializing a StringArray from this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable view, use discrete_string_variables_view())
• void discrete_string_variable(const String &ds_var, size_t index)
  set an active discrete string variable
• void discrete_string_variables(StringMultiArrayConstView ds_vars)
  set the active discrete string variables
• Real discrete_real_variable(size_t index) const
  return an active discrete real variable
• const RealVector & discrete_real_variables() const
  return the active discrete real variables (Note: returns a view by const reference, but initializing a RealVector from this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable view, use discrete_real_variables_view())
• void discrete_real_variable(Real dr_var, size_t index)
  set an active discrete real variable
• void discrete_real_variables(const RealVector &dr_vars)
  set the active discrete real variables
• void active_variables(const Variables &vars)
  copy the active cv/div/dsv/drv variables from vars
• void all_variables(const Variables &vars)
  copy all cv/div/dsv/drv variables from vars
• RealVector & continuous_variables_view()
  return a mutable view of the active continuous variables
• IntVector & discrete_int_variables_view()
return a mutable view of the active discrete integer variables

- StringMultiArrayView `discrete_string_variables_view()`
  return a mutable view of the active discrete string variables

- RealVector & `discrete_real_variables_view()`
  return a mutable view of the active discrete real variables

- StringMultiArrayConstView `continuous_variable_labels()` const
  return the active continuous variable labels

- void `continuous_variable_labels(StringMultiArrayConstView cv_labels)`
  set the active continuous variable labels

- void `continuous_variable_label(const String &cv_label, size_t index)`
  set an active continuous variable label

- StringMultiArrayConstView `discrete_int_variable_labels()` const
  return the active discrete integer variable labels

- void `discrete_int_variable_labels(StringMultiArrayConstView div_labels)`
  set the active discrete integer variable labels

- void `discrete_int_variable_label(const String &div_label, size_t index)`
  set an active discrete integer variable label

- StringMultiArrayConstView `discrete_string_variable_labels()` const
  return the active discrete string variable labels

- void `discrete_string_variable_labels(StringMultiArrayConstView dsv_labels)`
  set the active discrete string variable labels

- void `discrete_string_variable_label(const String &dsv_label, size_t index)`
  set an active discrete string variable label

- StringMultiArrayConstView `discrete_real_variable_labels()` const
  return the active discrete real variable labels

- void `discrete_real_variable_labels(StringMultiArrayConstView drv_labels)`
  set the active discrete real variable labels

- void `discrete_real_variable_label(const String &drv_label, size_t index)`
  set an active discrete real variable label

- UShortMultiArrayConstView `continuous_variable_types()` const
  return the active continuous variable types

- void `continuous_variable_types(UShortMultiArrayConstView cv_types)`
  set the active continuous variable types

- void `continuous_variable_type(unsigned short cv_type, size_t index)`
  set an active continuous variable type

- UShortMultiArrayConstView `discrete_int_variable_types()` const
  return the active discrete integer variable types

- void `discrete_int_variable_types(UShortMultiArrayConstView div_types)`
  set the active discrete integer variable types

- void `discrete_int_variable_type(unsigned short div_type, size_t index)`
  set an active discrete integer variable type

- UShortMultiArrayConstView `discrete_string_variable_types()` const
  return the active discrete string variable types
• void discrete_string_variable_types (UShortMultiArrayConstView dsv_types)
  set the active discrete string variable types
• void discrete_string_variable_type (unsigned short dsv_type, size_t index)
  set an active discrete string variable type
• UShortMultiArrayConstView discrete_real_variable_types () const
  return the active discrete real variable types
• void discrete_real_variable_types (UShortMultiArrayConstView drv_types)
  set the active discrete real variable types
• void discrete_real_variable_type (unsigned short drv_type, size_t index)
  set an active discrete real variable type
• SizetMultiArrayConstView continuous_variable_ids () const
  return the active continuous variable position identifiers
• void continuous_variable_ids (SizetMultiArrayConstView cv_ids)
  set the active continuous variable position identifiers
• void continuous_variable_id (size_t cv_id, size_t index)
  set an active continuous variable position identifier
• const RealVector & inactive_continuous_variables () const
  return the inactive continuous variables
• void inactive_continuous_variables (const RealVector &ic_vars)
  set the inactive continuous variables
• void inactive_continuous_variable (Real ic_var, size_t index)
  set an inactive continuous variable
• const IntVector & inactive_discrete_int_variables () const
  return the inactive discrete int variables
• void inactive_discrete_int_variables (const IntVector &idi_vars)
  set the inactive discrete int variables
• void inactive_discrete_int_variable (int idi_var, size_t index)
  set an inactive discrete int variable
• StringMultiArrayConstView inactive_discrete_string_variables () const
  return the inactive discrete string variables
• void inactive_discrete_string_variables (StringMultiArrayConstView ids_vars)
  set the inactive discrete string variables
• void inactive_discrete_string_variable (const String &ids_var, size_t index)
  set an inactive discrete string variable
• const RealVector & inactive_discrete_real_variables () const
  return the inactive discrete real variables
• void inactive_discrete_real_variables (const RealVector &idr_vars)
  set the inactive discrete real variables
• void inactive_discrete_real_variable (Real idr_var, size_t index)
  set an inactive discrete real variable
• StringMultiArrayConstView inactive_continuous_variable_labels () const
  return the inactive continuous variable labels
• void inactive_continuous_variable_labels (StringMultiArrayConstView ic_vars)
set the inactive continuous variable labels

- StringMultiArrayConstView inactive_discrete_int_variable_labels () const
  return the inactive discrete variable labels
- void inactive_discrete_int_variable_labels (StringMultiArrayConstView idi_vars)
  set the inactive discrete variable labels
- StringMultiArrayConstView inactive_discrete_string_variable_labels () const
  return the inactive discrete variable labels
- void inactive_discrete_string_variable_labels (StringMultiArrayConstView ids_vars)
  set the inactive discrete variable labels
- StringMultiArrayConstView inactive_discrete_real_variable_labels () const
  return the inactive discrete variable labels
- void inactive_discrete_real_variable_labels (StringMultiArrayConstView idr_vars)
  set the inactive discrete variable labels
- UShortMultiArrayConstView inactive_continuous_variable_types () const
  return the inactive continuous variable types
- UShortMultiArrayConstView inactive_discrete_int_variable_types () const
  return the inactive discrete integer variable types
- UShortMultiArrayConstView inactive_discrete_string_variable_types () const
  return the inactive discrete string variable types
- UShortMultiArrayConstView inactive_discrete_real_variable_types () const
  return the inactive discrete real variable types
- SizetMultiArrayConstView inactive_continuous_variable_ids () const
  return the inactive continuous variable position identifiers
- const RealVector & all_continuous_variables () const
  returns a single array with all continuous variables
- void all_continuous_variables (const RealVector &ac_vars)
  sets all continuous variables using a single array
- void all_continuous_variable (Real ac_var, size_t index)
  set a variable within the all continuous array
- const IntVector & all_discrete_int_variables () const
  returns a single array with all discrete variables
- void all_discrete_int_variables (const IntVector &adi_vars)
  sets all discrete variables using a single array
- void all_discrete_int_variable (int adi_var, size_t index)
  set a variable within the all discrete array
- StringMultiArrayConstView all_discrete_string_variables () const
  returns a single array with all discrete variables
- void all_discrete_string_variables (StringMultiArrayConstView ads_vars)
  sets all discrete variables using a single array
- void all_discrete_string_variable (const String &ads_var, size_t index)
  set a variable within the all discrete array
- const RealVector & all_discrete_real_variables () const
  returns a single array with all discrete variables
- void all_discrete_real_variables (const RealVector &adr_vars)
  sets all discrete variables using a single array
- void all_discrete_real_variable (Real adr_var, size_t index)
  set a variable within the all discrete array
- void as_vector (const StringSetArray &dss_vals, RealVector &var_values) const
  get the active variables as a vector of reals, converting string values to zero-based set indices
- StringMultiArrayView all_continuous_variable_labels () const
  returns a single array with all continuous variable labels
- void all_continuous_variable_labels (StringMultiArrayConstView acv_labels)
  sets all continuous variable labels using a single array
- void all_continuous_variable_label (const String &acv_label, size_t index)
  set a label within the all continuous label array
- StringMultiArrayView all_discrete_int_variable_labels () const
  returns a single array with all discrete variable labels
- void all_discrete_int_variable_labels (StringMultiArrayConstView adiv_labels)
  sets all discrete variable labels using a single array
- void all_discrete_int_variable_label (const String &adiv_label, size_t index)
  set a label within the all discrete label array
- StringMultiArrayView all_discrete_string_variable_labels () const
  returns a single array with all discrete variable labels
- void all_discrete_string_variable_labels (StringMultiArrayConstView adsv_labels)
  sets all discrete variable labels using a single array
- void all_discrete_string_variable_label (const String &adsv_label, size_t index)
  set a label within the all discrete label array
- UShortMultiArrayConstView all_continuous_variable_types () const
  return all continuous variable types
- UShortMultiArrayConstView all_discrete_int_variable_types () const
  return all discrete variable types
- UShortMultiArrayConstView all_discrete_string_variable_types () const
  return all discrete variable types
- UShortMultiArrayConstView all_discrete_real_variable_types () const
  return all discrete variable types
- SizetMultiArrayConstView all_continuous_variable_ids () const
  return all continuous variable position identifiers
- Variables copy (bool deep_svd=false) const
  a deep variables copy for use in history mechanisms (SharedVariablesData uses a shallow copy by default)
- const std::pair< short, short >& view () const
returns variablesView

- std::pair<short, short> get_view (const ProblemDescDB &problem_db) const
defines variablesView from problem_db attributes

- void inactive_view (short view2)
  sets the inactive view based on higher level (nested) context

- const String & variables_id () const
  returns the variables identifier string

- const SizetArray & variables_components_totals () const
  returns the number of variables for each of the constitutive components

- bool is_null () const
  function to check variablesRep (does this envelope contain a letter)

Protected Member Functions

- Variables (BaseConstructor, const ProblemDescDB &problem_db, const std::pair<short, short> &view)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- Variables (BaseConstructor, const SharedVariablesData &svd)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- void build_views ()
  construct active/inactive views of all variables arrays

- void build_active_views ()
  construct active views of all variables arrays

- void build_inactive_views ()
  construct inactive views of all variables arrays

Protected Attributes

- SharedVariablesData sharedVarsData
  reference-counted instance of shared variables data: id’s, labels, counts

- RealVector allContinuousVars
  array combining all of the continuous variables

- IntVector allDiscreteIntVars
  array combining all of the discrete integer variables

- StringMultiArray allDiscreteStringVars
  array combining all of the discrete string variables

- RealVector allDiscreteRealVars
  array combining all of the discrete real variables

- RealVector continuousVars
  the active continuous variables array view

- IntVector discreteIntVars
  the active discrete integer variables array view

- RealVector discreteRealVars
the active discrete real variables array view

- **RealVector** `inactiveContinuousVars`
  the inactive continuous variables array view

- **IntVector** `inactiveDiscreteIntVars`
  the inactive discrete integer variables array view

- **RealVector** `inactiveDiscreteRealVars`
  the inactive discrete real variables array view

### Private Member Functions

- **Variables** * get_variables (const ProblemDescDB &problem_db)
  Used by the standard envelope constructor to instantiate the correct letter class.

- **Variables** * get_variables (const SharedVariablesData &svd) const
  Used by the alternate envelope constructors, by read functions, and by `copy()` to instantiate a new letter class.

- **short method** `map` (short view_spec, bool relaxed) const
  infer domain from method selection

- **short method** `domain` (const ProblemDescDB &problem_db) const
  infer domain from method selection

- **short method** `view` (const ProblemDescDB &problem_db) const
  infer view from method selection

- **short response** `view` (const ProblemDescDB &problem_db) const
  infer view from type of response data set

- **void** `check_view_compatibility` ()
  perform sanity checks on view.first and view.second after update

- **template<class Archive>**
  void `load` (Archive &ar, const unsigned int version)
  read a Variables object from an archive

- **template<class Archive>**
  void `save` (Archive &ar, const unsigned int version) const
  write a Variables object to an archive

- **BOOST_SERIALIZATION_SPLIT_MEMBER ()** Variables *variablesRep
  pointer to the letter (initialized only for the envelope)

### Private Attributes

- **int** referenceCount
  number of objects sharing variablesRep

### Friends

- **class** boost::serialization::access
  for serializing private data members

- **bool** operator==(const Variables &vars1, const Variables &vars2)
  strict equality operator (for boost hash-based lookups)

- **bool** operator!=(const Variables &vars1, const Variables &vars2)
13.219.1 Detailed Description

Base class for the variables class hierarchy.

The Variables class is the base class for the class hierarchy providing design, uncertain, and state variables for continuous and discrete domains within a Model. Using the fundamental arrays from the input specification, different derived classes define different views of the data. For memory efficiency and enhanced polymorphism, the variables hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Variables) serves as the envelope and one of the derived classes (selected in Variables::get_variables()) serves as the letter.

13.219.2 Member Function Documentation

StringMultiArrayView discrete_string_variables_view () [inline]

return a mutable view of the active discrete string variables
same as discrete_string_variables(), except mutable view
References Variables::allDiscreteStringVars, SharedVariablesData::dsv(), SharedVariablesData::ds_start(), and Variables::sharedVarsData.
The documentation for this class was generated from the following file:

- DakotaVariables.hpp

13.220 Verification Class Reference

Base class for managing common aspects of verification studies.

Inheritance diagram for Verification:

```
  RichExtrapVerification
    Verification
    Analyzer
    Iterator
```

Public Member Functions

- bool resize ()

  reinitializes iterator based on new variable size
Protected Member Functions

- **Verification** (ProblemDescDB &problem_db, Model &model)
  constructor

- **Verification** (unsigned short method_name, Model &model)
  alternate constructor for instantiations "on the fly"

- ~Verification ()
  destructor

- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results

Additional Inherited Members

13.220.1 Detailed Description

Base class for managing common aspects of verification studies.

The Verification base class manages common data and functions, such as those involving ...

13.220.2 Member Function Documentation

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Analyzer.
Reimplemented in RichExtrapVerification.
References Analyzer::print_results().
Referenced by RichExtrapVerification::print_results().
The documentation for this class was generated from the following files:

- DakotaVerification.hpp
- DakotaVerification.cpp

13.221 VLint Struct Reference

structure for validating integer uncertain variable labels, bounds, values

Public Attributes

- int n
- VarLabel Var_Info::* VL
- Var_uinfo * vui
- StringArray DataVariablesRep::* Labels
- IntVector DataVariablesRep::* LowerBnds
- IntVector DataVariablesRep::* UpperBnds
- IntVector DataVariablesRep::* UncVars
13.221.1 Detailed Description
structure for validating integer uncertain variable labels, bounds, values
The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

13.222 VLreal Struct Reference
structure for validating real uncertain variable labels, bounds, values

Public Attributes
- int n
- VarLabel Var_Info::* VL
- Var_uinfo * vui
- StringArray DataVariablesRep::* Labels
- RealVector DataVariablesRep::* LowerBnds
- RealVector DataVariablesRep::* UpperBnds
- RealVector DataVariablesRep::* UncVars

13.222.1 Detailed Description
structure for validating real uncertain variable labels, bounds, values
The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

13.223 VLstr Struct Reference
structure for validating string uncertain variable labels, bounds, values

Public Attributes
- int n
- VarLabel Var_Info::* VL
- Var_uinfo * vui
- StringArray DataVariablesRep::* Labels
- StringArray DataVariablesRep::* LowerBnds
- StringArray DataVariablesRep::* UpperBnds
- StringArray DataVariablesRep::* UncVars

13.223.1 Detailed Description
structure for validating string uncertain variable labels, bounds, values
The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp
13.224 VPSApproximation Class Reference

Derived approximation class for VPS implementation.

Inheritance diagram for VPSApproximation:

```
<table>
<thead>
<tr>
<th>Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPSApproximation</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **VPSApproximation ()**
  
  *default constructor*

- **VPSApproximation (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  *standard constructor (to call VPS from an input deck)*

- **VPSApproximation (const SharedApproxData &shared_data)**
  
  *Alternate constructor (to call VPS from another method like POF-darts)*

- **~VPSApproximation ()**
  
  *destructor*

- **bool VPS_execute ()**
- **void VPS_create_containers ()**
- **void VPS_retrieve_neighbors (size_t ipoint, bool update_point_neighbors)**
- **void VPS_adjust_extend_neighbors_of_all_points ()**
- **void VPS_extend_neighbors (size_t ipoint)**
- **void VPS_build_local_surrogate (size_t cell_index)**
- **double VPS_evaluate_surrogate (double *x)**
- **void VPS_destroy_global_containers ()**
- **void retrieve_permutations (size_t &m, size_t **perm, size_t num_dim, size_t upper_bound, bool force_sum_constraint, size_t sum_constraint)**
- **void build_radial_basis_function (size_t icell)**
- **void VPS_LS_retrieve_weights (size_t cell_index)**
- **double evaluate_basis_function (double *x, size_t icell, size_t ibasis)**
- **int constrained_LeastSquare (size_t n, size_t m, double **H, double *w, double *f)**
- **double vec_dot_vec (size_t n, double *vec_a, double *vec_b)**
- **double vec_pow_vec (size_t num_dim, double *vec_a, size_t *vec_b)**
- **bool Cholesky (int n, double **A, double **LD)**
- **void Cholesky_solver (int n, double **LD, double *b, double *x)**
- **void GMRES (size_t n, double **A, double *b, double *x, double eps)**
- **void printMatrix (size_t m, size_t n, double **M)**
- **void initiate_random_number_generator (unsigned long x)**
- **double generate_a_random_number ()**
- **size_t retrieve_closest_cell (double *x)**
- **bool trim_line_using_Hyperplane (size_t num_dim, double *st, double *end, double *qH, double *nH)**
• double f_test (double *x)
• double * grad_f_test (double *x)
• double ** hessian_f_test (double *x)
• void generate_poisson_disk_sample (double r)
• void generate_MC_sample ()
• void isocontouring (std::string file_name, bool plot_test_function, bool plot_surrogate, std::vector< double > contours)
• void isocontouring_solid (std::string file_name, bool plot_test_function, bool plot_surrogate, std::vector< double > contours)
• void plot_neighbors ()

Protected Member Functions

• int min_coefficients () const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions
• int num_constraints () const
  return the number of constraints to be enforced via an anchor point
• void build (size_t index=NPOS)
  builds the approximation from scratch
• Real value (const Variables &vars)
  retrieve the predicted function value for a given parameter set
• const RealVector & gradient (const Variables &vars)
  retrieve the function gradient at the predicted value for a given parameter set
• Real prediction_variance (const Variables &vars)
  retrieve the variance of the predicted value for a given parameter set

Private Types

• enum subsurrogate { LS, GP }
• enum subsurrogate_basis { polynomial, radial }
• enum testfunction {
  SmoothHerbie, Herbie, Cone, Cross,
  UnitSphere, Linear34 }

Private Member Functions

• void VPSmodel_build ()
  Function to compute coefficients governing the VPS surrogates.
• void VPSmodel_apply (const RealVector &new_x, bool variance_flag, bool gradients_flag)
  Function returns a response value using the VPS surface.
Private Attributes

- Real \texttt{approxValue}
  \textit{value of the approximation returned by \texttt{value}()}
- Real \texttt{approxVariance}
  \textit{value of the approximation returned by \texttt{prediction_variance}()} 
- \texttt{RealMatrix trainPoints}
  \textit{A 2-D array (num sample sites = rows, num vars = columns) used to create the Gaussian process.}
- \texttt{RealMatrix trainValues}
  \textit{An array of response values; one response value per sample site.}
- \texttt{size_t numObs}
  \textit{The number of observations on which the GP surface is built.}
- \texttt{int surrogateOrder}
  \textit{The order of the polynomial in each Voronoi cell.}
- \texttt{subsurrogate \_vps\_subsurrogate}
- \texttt{subsurrogate\_basis \_vps\_subsurrogate\_basis}
- \texttt{testfunction \_vps\_test\_function}
- \texttt{double Q [1220]}
- \texttt{int indx}
- \texttt{double cc}
- \texttt{double c}
- \texttt{double zc}
- \texttt{double zx}
- \texttt{double zy}
- \texttt{size_t qlen}
- \texttt{size_t \_n_dim}
- \texttt{double \_xmin}
- \texttt{double \_xmax}
- \texttt{double \_diag}
- \texttt{size_t \_num_inserted_points}
- \texttt{double ** \_sample_points}
- \texttt{double \_fval}
- \texttt{double ** \_fgrad}
- \texttt{double *** \_fhes}
- \texttt{size_t ** \_sample_neighbors}
- \texttt{size_t ** \_vps\_ext\_neighbors}
- \texttt{size_t \_vps\_order}
- \texttt{size_t \_num\_GMRES}
- \texttt{size_t \_num\_cell\_basis\_functions}
- \texttt{double ** \_sample\_vsize}
- \texttt{double ** \_vps\_dfar}
- \texttt{double *** \_sample\_basis}
- \texttt{double \_max\_vsize}
- \texttt{double \_disc\_min\_jump}
- \texttt{double \_disc\_min\_grad}
- \texttt{double \_f\_min}
13.224. VPSAPPROXIMATION CLASS REFERENCE

- double _f_max
- size_t *** _vps_t
- double ** _vps_w
- SharedApproxData sharedData
- std::vector< Approximation > gpApproximations
- Variables gpEvalVars
- bool _use_derivatives
- bool _use_gradient
- bool _use_hessian

Static Private Attributes

- static VPSApproximation * VPSinstance

  pointer to the active object instance used within the static evaluator

Additional Inherited Members

13.224.1 Detailed Description

Derived approximation class for VPS implementation.

The VPSApproximation class provides a set of piecewise surrogate approximations each of which is valid within a Voronoi cell.

13.224.2 Member Function Documentation

void VPSmodel_apply ( const RealVector & new_x, bool variance_flag, bool gradients_flag ) [private]

Function returns a response value using the VPS surface.

  The response value is computed at the design point specified by the RealVector function argument.
  References VPSApproximation::approxValue.
  Referenced by VPSApproximation::gradient(), VPSApproximation::prediction_variance(), and VPSApproximation::value().

13.224.3 Member Data Documentation

VPSApproximation * VPSinstance [static], [private]

pointer to the active object instance used within the static evaluator

default constructor

The documentation for this class was generated from the following files:

- VPSApproximation.hpp
- VPSApproximation.cpp
13.225 WorkdirHelper Class Reference

Static Public Member Functions

- static void initialize ()
  
  initialize (at runtime) cached values for paths and environment

- static const std::string & startup_pwd ()

  Query for dakota’s startup SPWD.

- static void change_directory (const bfs::path &new_dir)

  change current directory

- static void prepend_preferred_env_path (const std::string &extra_path)

  Prepend cached preferredEnvPath with extra_path and update $PATH environment variable.

- static void set_environment (const std::string &env_name, const std::string &env_val, bool overwrite_flag=true)

  Set an environment variable.

- static bfs::path which (const std::string &driver_name)

  Returns the bfs::path for the analysis driver, supporting typical windows filename extensions, or empty if not found.

- static bfs::path rel_to_abs (const bfs::path &subdir_path)

  get a valid absolute bfs::path to a subdirectory relative to rundir

- static StringArray tokenize_driver (const String &user_an_driver)

  tokenize a white-space separated analysis driver, respecting escapes and nested quotes

- static bool resolve_driver_path (String &an_driver)

  parse off the first whitespace-separated entry in the user’s analysis_driver, and convert it to an absolute path if it begins with ./ or ../, replacing the passed string if needed. Returns true if the first token was modified.

- static void split_wildcard (const std::string &path with wc, bfs::path &search_dir, bfs::path &wildcard)

  given a string with an optional path and a wildcard, e.g., /tmp/D*.*pp, parse it into the search path /tmp (default .) and the wildcard D*.*pp. Return wildcard as path to reduce wstring conversions

- static bfs::path concat_path (const bfs::path &p_in, const String &tag)

  concatenate a string onto the end of a path

- static bfs::path system_tmp_file (const std::string &prefix)

  generate a valid temporary file name <prefix>.

- static bfs::path system_tmp_path ()

  get the system tmp path, e.g., /tmp or C:

- static bool create_directory (const bfs::path &dir_path, short mkdir_option)

  Create a directory, with options for remove or error.

- static void recursive_remove (const bfs::path &rm_path, short fileop_option)

  Remove a path (file, directory, or symlink) without regard to its type. Only error if existed and there’s an error in the remove.

- static void rename (const bfs::path &old_path, const bfs::path &new_path, short fileop_option)

  Rename a file, catching any errors and optionally warning/erroring.

- static void link_items (const StringArray &source_items, const bfs::path &dest_dir, bool overwrite)

  top-level link a list of source paths (files, directories, symlinks), potentially including wildcards, from destination_dir, which must exist

- static void copy_items (const StringArray &source_items, const bfs::path &dest_dir, bool overwrite)

  copy a list of source paths (files, directories, symlinks), potentially including wildcards into destination_dir, which must exist
• static void prepend_path_items (const StringArray &source_items)
  prepend any directories (including wildcards) found in source_items to the preferred environment path; this will
  update cached preferred path and PATH
• static bool check_equivalent_dest (const StringArray &source_items, const bfs::path &dest_dir)
  check whether any of the passed source items are filesystem equivalent to the destination path, return true if any
  one is equivalent to dest
• static bool find_driver (const StringArray &source_items, const bfs::path &search_driver)
  check whether the any of the passed source items (possibly including wildcards to be expanded) matches the passed
  search driver
• static bool link (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  create link from dest_dir/src_path.filename() to a single path (file, dir, link) in source directory
• static bool recursive_copy (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  Recursive copy of src_path into dest_dir, with optional top-level overwrite (remove/recreate) of dest_dir/src_path.-
  filename()
• static bool prepend_path_item (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  prepend the preferred env path with source path if it's a directory; this will update cached preferred path and
  manipulate PATH
• static bool check_equivalent (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  return true if the src and dest are filesystem equivalent
• static bool find_file (const bfs::path &src_path, const bfs::path &search_file, bool overwrite)
  return true if the src_path is a regular file and has same filename as search_file
• static bool file_op_items (const file_op_function &file_op, const StringArray &source_paths, const bfs::path
  &dest_dir, bool overwrite)
  recursively perform file_op (copy, path adjust, etc.) on a list of source_paths (files, directories, symlinks), which
  potentially include wildcards, w.r.t. destination_dir
• static void set_preferred_path ()
  set/reset PATH to dakPreferredEnvPath
• static void set_preferred_path (const boost::filesystem::path &extra_path)
  set PATH to absolute(extra_path):dakPreferredEnvPath, without changing cached preferred PATH
• static void reset ()
  Resets the working directory "state" to its initial state when DAKOTA was launched.

Private Member Functions

• WorkdirHelper ()
  default constructor
• WorkdirHelper (const WorkdirHelper &)
  copy constructor
• ~WorkdirHelper ()
  destructor
• WorkdirHelper & operator= (const WorkdirHelper &)
  assignment operator
Static Private Member Functions

- static bfs::path po_which (const std::string &driver_name)
  Returns the bfs::path for the analysis driver - POSIX-style implementation, returns empty if not found.

- static std::string init_startup_path ()
  Initializes class member, startupPATH.

- static std::string init_preferred_env_path ()
  Initializes class member, dakPreferredEnvPath.

- static std::vector<std::string> tokenize_env_path (const std::string &path)
  Tokenizes $PATH environment variable into a "list" of directories.

Static Private Attributes

- static std::string startupPWD = ".";
  Value of $PWD var upon entry to dakota main()

- static std::string startupPATH = ".";
  Value of $PATH (PATH% on windows) var upon entry to dakota main(), omitting any leading PATH= or Path=.

- static std::string dakPreferredEnvPath = ".";
  Dakota preferred search PATH/Path = ":startupPWD:startupPATH", omitting any leading PATH= or Path=.

13.225.1 Detailed Description

Utility class for cross-platform management of environment and paths. Including directory and file operations. On initialization, this class does not manipulate the present working directory, nor the PATH environment variable, but stores context to manipulate them later.

13.225.2 Member Function Documentation

void initialize ( ) [static]

Initialize (at runtime) cached values for paths and environment.
Initialize defers calls to Boost filesystem utilities until runtime (required on some operating systems).
References WorkdirHelper::dakPreferredEnvPath, WorkdirHelper::init_preferred_env_path(), WorkdirHelper::init_startup_path(), WorkdirHelper::startupPATH, and WorkdirHelper::startupPWD.
Referenced by Environment::Environment().

void prepend_preferred_env_path ( const std::string &extra_path ) [static]

Prepend cached preferredEnvPath with extra_path and update $PATH environment variable.
Overwrites $PATH with an additional directory prepended, typically for the purpose of ensuring templatedir is in the $PATH; updates cached preferred PATH and environment PATH, so exercise caution with repeated calls.
References WorkdirHelper::dakPreferredEnvPath, WorkdirHelper::set_environment(), and WorkdirHelper::startupPWD.
Referenced by WorkdirHelper::prepend_path_item().
**bfs::path which ( const std::string & driver_name ) [static]**

Returns the bfs::path for the analysis driver, supporting typical windows filename extensions, or empty if not found.

Uses string representing $PATH to locate an analysis driver on the host computer. Returns the path to the driver (as a string).

This version is a wrapper over the "plain ol' which" implementation, allowing an array of windows, 3-letter extensions to be checked.

References Dakota::get_pathext(), and WorkdirHelper::po_which().

Referenced by NIDRProblemDescDB::check_driver().

**void splitWildcard ( const std::string & path_with_wc, bfs::path & search_dir, bfs::path & wildcard ) [static]**

given a string with an optional path and a wildcard, e.g., /tmp/D*.?pp, parse it into the search path /tmp (default .) and the wildcard D*.?pp. Return wildcard as path to reduce wstring conversions

Input: path_with_wc; Output: search_dir, wildcard

Referenced by WorkdirHelper::file_op_items().

**bool create_directory ( const bfs::path & dir_path, short mkdir_option ) [static]**

Create a directory, with options for remove or error.

mkdir_option is DIR_CLEAN (remove and recreate), DIR_PERSIST (leave existing), or DIR_ERROR (don't allow existing) returns whether a new directory was created.

References Dakota::abort_handler(), and WorkdirHelper::recursive_remove().

Referenced by ProcessApplicInterface::define_filenames().

**void link_items ( const StringArray & source_items, const bfs::path & dest_dir, bool overwrite ) [static]**

top-level link a list of source_paths (files, directories, symlinks), potentially including wildcards, from destination_dir, which must exist

Iterate source items (paths or wildcards), linking each of them from the destination. If overwrite, remove and replace any existing destination target, otherwise, allow to persist

References WorkdirHelper::file_op_items(), and WorkdirHelper::link().

Referenced by ProcessApplicInterface::define_filenames().

**void copy_items ( const StringArray & source_items, const bfs::path & dest_dir, bool overwrite ) [static]**

copy a list of source_paths (files, directories, symlinks), potentially including wildcards into destination_dir, which must exist

Iterate source items (paths or wildcards), copying each of them into the destination. If overwrite, remove and replace any existing destination target, otherwise, allow to persist

References WorkdirHelper::file_op_items(), and WorkdirHelper::recursive_copy().

Referenced by ProcessApplicInterface::define_filenames().
bool link (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite) [static]

create link from dest_dir/src_path.filename() to a single path (file, dir, link) in source directory
  Assumes source file exists since it was iterated in the calling context. If overwrite, any existing file in dest_dir will be removed prior to creating the new link.
  References Dakota::abort_handler().
  Referenced by WorkdirHelper::link_items().

bool recursive_copy (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite) [static]

Recursive copy of src_path into dest_dir, with optional top-level overwrite (remove/recreate) of dest_dir/src_path.filename()
  note dest_dir is the containing folder for the src_path contents to be placed in for consistency with other convenience functions (may need to reconsider)
  References Dakota::abort_handler().
  Referenced by WorkdirHelper::copy_items().

bool prepend_path_item (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite) [static]

prepend the preferred env path with source path if it’s a directory; this will update cached preferred path and manipulate PATH
  prepend the env path with source path if it’s a directory or directory symlink
  References Dakota::abort_handler(), and WorkdirHelper::prepend_preferred_env_path().
  Referenced by WorkdirHelper::prepend_path_items().

bool file_op_items (const file_op_function &file_op, const StringArray &source_items, const bfs::path &dest_dir, bool overwrite) [static]

recursively perform file_op (copy, path adjust, etc.) on a list of source_paths (files, directories, symlinks), which potentially include wildcards, w.r.t. destination_dir
  Iterator implementation for copy, link, etc file operation. Iterate source items (paths or wildcards), performing file_op on each w.r.t. destination. If overwrite, remove and replace any existing destination target (at top-level), otherwise, allow to persist. Return code true indicates abnormal behavior.
  References WorkdirHelper::split_wildcard(), and Dakota::strcontains().
  Referenced by WorkdirHelper::check_equivalent_dest(), WorkdirHelper::copy_items(), WorkdirHelper::find_-driver(), WorkdirHelper::link_items(), and WorkdirHelper::prepend_path_items().

void set_preferred_path (const boost::filesystem::path &extra_path) [static]

set PATH to absolute(extra_path):dakPreferredEnvPath, without changing cached preferred PATH
  If needed, convert the passed item to an absolute path (while could make sense to prepend a relative path, no current use cases) and prepend when setting environment. Does not update cached preferred path.
  References WorkdirHelper::dakPreferredEnvPath, WorkdirHelper::rel_to_abs(), and WorkdirHelper::set_environment().

bfs::path po_which (const std::string &driver_name) [static], [private]

Returns the bfs::path for the analysis driver - POSIX-style implementation, returns empty if not found.
  For absolute driver_name, validates that is regular file. For relative, uses string representing $PATH (preferred path) to locate an analysis driver on the host computer. Returns the path to the driver, or empty if not found.
  This is the "plain ol’ which" impl that worked well, historically, on POSIX.
References Dakota::contains(), WorkdirHelper::dakPreferredEnvPath, and WorkdirHelper::tokenize_env_path(). Referenced by WorkdirHelper::which().

std::string init_startup_path() [static], [private]

Initializes class member, startupPATH.
- Gets the $PATH (PATH% on windows) and returns the std::string value
- References Dakota::abort_handler().
- Referenced by WorkdirHelper::initialize().

std::string init_preferred_env_path() [static], [private]

Initializes class member, dakPreferredEnvPath.
- Prepends `.` and the startupPWD to the initial startup $PATH string so that analysis driver detection is more robust
- References WorkdirHelper::startupPATH, and WorkdirHelper::startupPWD.
- Referenced by WorkdirHelper::initialize().

std::vector<std::string> tokenize_env_path(const std::string &env_path) [static], [private]

Tokenizes $PATH environment variable into a "list" of directories.
- Creates a a vector of directories (as an aid to search) by breaking up the $PATH environment variable (passed in as a string argument)
- Referenced by WorkdirHelper::po_which().
- The documentation for this class was generated from the following files:
  - WorkdirHelper.hpp
  - WorkdirHelper.cpp
Chapter 14

File Documentation

14.1 dakota_dll_api.cpp File Reference

This file contains a DakotaRunner class, which launches DAKOTA.

Namespaces

- Dakota
  
  The primary namespace for DAKOTA.

Constant Groups

- Dakota
  
  The primary namespace for DAKOTA.

Functions

- void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, const char *logname)
  
  create and configure a new DakotaRunner, adding it to list of instances

- int DAKOTA_DLL_FN dakota_readInput (int id, const char *dakotaInput)
  
  command DakotaRunner instance id to read from file dakotaInput

- void DAKOTA_DLL_FN dakota_get_variable_info (int id, char ***pVarNames, int *pNumVarNames, char ***pRespNames, int *pNumRespNames)
  
  return the variable and response names

- int DAKOTA_DLL_FN dakota_start (int id)
  
  command DakotaRunner instance id to start (plugin interface and run strategy)

- void DAKOTA_DLL_FN dakota_destroy (int id)
  
  delete Dakota runner instance id and remove from active list

- void DAKOTA_DLL_FN dakota_stop (int *id)
  
  command DakotaRunner instance id to stop execution

- const char *DAKOTA_DLL_FN dakota_getStatus (int id)
  
  return current results output as a string

- int get_me_ptr_int ()
get the DAKOTA pointer to ModelCenter
- void set_mc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter
- int get_dc_ptr_int ()
  get the DAKOTA pointer to ModelCenter current design point
- void set_dc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter current design point

14.1.1 Detailed Description
This file contains a DakotaRunner class, which launches DAKOTA.

14.1.2 Function Documentation

void DAKOTA_DLL_FN dakota_stop ( int *id )
command DakotaRunner instance id to stop execution
  TODO: trick application to quit through the syscall interface or throw exception.

14.2 dakota_dll_api.h File Reference

API for DLL interactions.

Functions
- void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, const char *logname)
  create and configure a new DakotaRunner, adding it to list of instances
- int DAKOTA_DLL_FN dakota_readInput (int id, const char *dakotaInput)
  command DakotaRunner instance id to read from file dakotaInput
- int DAKOTA_DLL_FN dakota_start (int id)
  command DakotaRunner instance id to start (plugin interface and run strategy)
- void DAKOTA_DLL_FN dakota_destroy (int id)
  delete Dakota runner instance id and remove from active list
- void DAKOTA_DLL_FN dakota_stop (int *id)
  command DakotaRunner instance id to stop execution
- const char *DAKOTA_DLL_FN dakota_getStatus (int id)
  return current results output as a string
- int DAKOTA_DLL_FN get_mc_ptr_int ()
  get the DAKOTA pointer to ModelCenter
- void DAKOTA_DLL_FN set_mc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter
- int DAKOTA_DLL_FN get_dc_ptr_int ()
  get the DAKOTA pointer to ModelCenter current design point
- void DAKOTA_DLL_FN set_dc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter current design point
14.3 DAKOTA_LINEAR_ALGEBRA.HPP FILE REFERENCE

- void DAKOTA_DLL_FN dakota_get_variable_info (int id, char **pVarNames, int *pNumVarNames, char **pRespNames, int *pNumRespNames)
  return the variable and response names

14.2.1 Detailed Description
API for DLL interactions.

14.2.2 Function Documentation
void DAKOTA_DLL_FN dakota_stop ( int *id )
command DakotaRunner instance id to stop execution
  TODO: trick application to quit through the syscall interface or throw exception.

14.3 dakota_linear_algebra.hpp File Reference
Dakota linear algebra utilities.

Namespaces
- Dakota
  The primary namespace for DAKOTA.

Constant Groups
- Dakota
  The primary namespace for DAKOTA.

Functions
- void svd (RealMatrix &matrix, RealVector &singular_vals, RealMatrix &v_trans, bool compute_vectors=true)
  Compute the SVD of an arbitrary matrix A = U S V^T.
- void singular_values (RealMatrix &matrix, RealVector &singular_values)
  compute the singular values without storing any singular vectors (A will be destroyed)
- int qr (RealMatrix &A)
  Compute an in-place QR factorization A = QR.
- int qr_rsolve (const RealMatrix &q_r, bool transpose, RealMatrix &rhs)
  Perform a multiple right-hand sides Rinv * rhs solve using the R from a qr factorization.
- double det_AtransA (RealMatrix &A)
  Use SVD to compute det(A'*A), destroying A with the SVD.

14.3.1 Detailed Description
Dakota linear algebra utilities. Convenience functions to perform Teuchos::LAPACK operations on Dakota RealMatrix/RealVector.
Utility functions for reading and writing tabular data files. Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data-util.h.

Namespaces

- **Dakota**
  
  *The primary namespace for DAKOTA.*

Constant Groups

- **Dakota**
  
  *The primary namespace for DAKOTA.*

Functions

- **String format_name** (unsigned short tabular_format)
  
  *Translate tabular_format into a user-friendly name.*

- **void print_expected_format** (std::ostream &s, unsigned short tabular_format, size_t num_rows, size_t num_cols)
  
  *Describe the expected data file format based on passed parameters.*

- **void print_unexpected_data** (std::ostream &s, const String &filename, const String &context_message, unsigned short tabular_format)
  
  *Print a warning if there’s extra data in the file.*

- **void open_file** (std::ifstream &data_file, const std::string &input_filename, const std::string &context_message)
  
  *Open the file specified by name for reading, using passed input stream, presenting context-specific error on failure.*

- **void open_file** (std::ofstream &data_file, const std::string &output_filename, const std::string &context_message)
  
  *Open the file specified by name for writing, using passed output stream, presenting context-specific error on failure.*

- **void close_file** (std::ifstream &data_file, const std::string &input_filename, const std::string &context_message)
  
  *Close the file specified by name after reading, using passed input stream, presenting context-specific error on failure.*

- **void close_file** (std::ofstream &data_file, const std::string &output_filename, const std::string &context_message)
  
  *Close the file specified by name after writing, using passed output stream, presenting context-specific error on failure.*

- **void write_header_tabular** (std::ostream &tabular_ostream, const Variables &vars, const Response &response, const std::string &counter_label, unsigned short tabular_format)
  
  *Output the header row (labels) for a tabular data file for variables and responses, with variables in input spec order. Conditionally include interface ID. Primary uses: environment tabular data, pre-run output, surrogate approx evals.*

- **void write_header_tabular** (std::ostream &tabular_ostream, const Variables &vars, const StringArray &addtnl_labels, const std::string &counter_label, unsigned short tabular_format)
Output the header row (labels) for a tabular data file for variables and additional labels not tied to a response. Variables are in input spec order. Conditionally include interface ID. Primary uses: MCMC chain export, including calibration sigmas.

- void \texttt{write\_leading\_columns} (std::ostream &tabular\_ostream, size\_t eval\_id, const String &iface\_id, unsigned short tabular\_format)
  
  Write the leading column with eval ID and conditionally, the interface ID.

- void \texttt{write\_data\_tabular} (std::ostream &tabular\_ostream, const Variables &vars, const String &iface, size\_t counter, unsigned short tabular\_format)
  
  Output a row of tabular data from a variables object. All active/inactive variables written in input spec order. Conditionally include interface ID. Primary uses: output of sampling sets.

- void \texttt{write\_data\_tabular} (std::ostream &tabular\_ostream, const Variables &vars, const String &iface, const Response &response, size\_t counter, unsigned short tabular\_format)
  
  Output a row of tabular data from variables and response objects. All active/inactive variables written in input spec order. Conditionally include interface ID. Primary uses: environment tabular data, pre-run output, surrogate approx evals.

- void \texttt{write\_data\_tabular} (const std::string &output\_filename, const std::string &context\_message, const RealVectorArray &output\_coeffs, const UShort2DArray &output\_indices)
  
  PCE export: write freeform format file with whitespace-separated data where each row has num\_fns reals from coeffs, followed by num\_vars unsigned shorts from indices.

- bool \texttt{exists\_extra\_data} (std::istream &tabular\_file)
  
  Check if an input stream contains unexpected additional data.

- void \texttt{read\_header\_tabular} (std::istream &input\_stream, unsigned short tabular\_format)
  
  read and discard header line from the stream

- int \texttt{read\_leading\_columns} (std::istream &input\_stream, unsigned short tabular\_format)
  
  read leading columns \[ int eval\_id [ String iface\_id ] \]

- void \texttt{read\_leading\_columns} (std::istream &input\_stream, unsigned short tabular\_format, int &eval\_id, String &iface\_id)
  
  read leading columns \[ int eval\_id [ String iface\_id ] \]

- void \texttt{read\_data\_tabular} (const std::string &input\_filename, const std::string &context\_message, RealVector &input\_data, size\_t num\_entries, unsigned short tabular\_format)
  
  read possibly header-annotated whitespace-separated data into a vector of length num\_entries; if annotated then it's a column vector for now

- void \texttt{read\_data\_tabular} (const std::string &input\_filename, const std::string &context\_message, Variables vars, size\_t num\_fns, RealMatrix &vars\_matrix, RealMatrix &resp\_matrix, unsigned short tabular\_format, bool active\_only)
  
  Tabular read for \texttt{ApproximationInterface} challenge data: read possibly header-annotated whitespace-separated data of possible mixed Variables, followed by num\_fns, each into RealMatrix with minimal error checking.

- void \texttt{read\_data\_tabular} (const std::string &input\_filename, const std::string &context\_message, RealVectorArray &input\_coeffs, UShort2DArray &input\_indices, unsigned short tabular\_format, size\_t num\_vars, size\_t num\_fns)
  
  Tabular read for PCE import: read possibly header-annotated whitespace-separated data of unknown length where each row has num\_fns reals followed by num\_vars unsigned shorts; append data to arrays passed by reference.

- void \texttt{read\_data\_tabular} (const std::string &input\_filename, const std::string &context\_message, Variables vars, Response resp, PRPList &input\_prp, unsigned short tabular\_format, bool verbose=false, bool active\_only=false)
  
  Tabular read for \texttt{DataFitSurrModel} (build points): read whitespace-separated data with optional row and column headers into lists of Variables and Responses until out of data.
- void read_data_tabular(const std::string &input_filename, const std::string &context_message, RealMatrix &input_matrix, size_t record_len, unsigned short tabular_format, bool verbose=false)
  
  Tabular read for import_approx_points_file: read whitespace-separated data with optional row and column headers into a single matrix, with length of record as specified and number of records to be determined by file content. The matrix is stored as record_len rows by num_records columns.

- void read_data_tabular(const std::string &input_filename, const std::string &context_message, RealMatrix &input_matrix, size_t num_rows, size_t num_cols, unsigned short tabular_format, bool verbose=false)
  
  Tabular read for GPMSA data: read whitespace-separated data with optional row and column headers into a single matrix, with size as specified (one experiment per row)

- size_t read_data_tabular(const std::string &input_filename, const std::string &context_message, RealVectorArray &cva, IntVectorArray &diva, StringMulti2DArray &dsva, RealVectorArray &drva, unsigned short tabular_format, bool active_only, Variables vars)
  
  Tabular read for ParamStudy: read specified input data file into arrays with sizes specified by the passed vc_totals array.

### 14.4.1 Detailed Description

Utility functions for reading and writing tabular data files Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data-util.h. Design/capability goals: Ability to read / write data with row/col headers or in free-form Detect premature end of file, report if extra data More consistent and reliable checks for file open errors Require right number of cols in header mode; only total data checking in free-form (likely) Allow comment character for header rows or even in data? variables vs. variables/responses for both read and write Should we support CSV? delimiter = ','. other? Verify treatment of trailing newline without reading a zero Allow reading into the transpose of the data structure

### 14.5 dll_tester.cpp File Reference

Test the DLL with a DAKOTA input file.

**Functions**

- int main (int argc, char ∗argv[ ])

  The main program for exercising the DLL API with a simple command-line.

### 14.5.1 Detailed Description

Test the DLL with a DAKOTA input file.

### 14.6 JEGAOptimizer.cpp File Reference

Contains the implementation of the JEGAOptimizer class.

**Classes**

- class JEGAOptimizer::Evaluator

  An evaluator specialization that knows how to interact with Dakota.
14.7. JEGAOPTIMIZER.HPP FILE REFERENCE

- **class** JEGAOptimizer::EvaluatorCreator
  
  A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

- **class** JEGAOptimizer::Driver
  
  A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.

**Namespaces**

- **Dakota**

  *The primary namespace for DAKOTA.*

**Constant Groups**

- **Dakota**

  *The primary namespace for DAKOTA.*

**Functions**

- template<typename T>
  
  string `asstring` (const T &val)

  *Creates a string from the argument val using an ostringstream.*

14.6.1 Detailed Description

Contains the implementation of the JEGAOptimizer class.

14.7 JEGAOptimizer.hpp File Reference

Contains the definition of the JEGAOptimizer class.

**Classes**

- **class** JEGAOptimizer

  *A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).*

- **class** JEGATraits

  *A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).*

**Namespaces**

- **Dakota**

  *The primary namespace for DAKOTA.*

**Constant Groups**

- **Dakota**

  *The primary namespace for DAKOTA.*
14.7.1 Detailed Description
Contains the definition of the JEGAOptimizer class.

14.8 library_mode.cpp File Reference
file containing a mock simulator main for testing Dakota in library mode

Classes
- struct callback_data

Functions
- void fpinit_AS ()
- void run_dakota_parse (const char *dakota_input_file)
  Run a Dakota LibraryEnvironment, mode 1: parsing an input file.
- void run_dakota_data ()
  Run a Dakota LibraryEnvironment, mode 2: from C++ API inserted data.
- void run_dakota_mixed (const char *dakota_input_file, bool mpirun_flag)
  Run a Dakota LibraryEnvironment, from string or input file input, supplemented with additional C++ API adjustments.
- void serial_interface_plugin (Dakota::LibraryEnvironment &env)
  Convenience function with simplest example of interface plugin: plugin a serial DirectApplicInterface that can be constructed independent of Dakota’s configuration details.
- void parallel_interface_plugin (Dakota::LibraryEnvironment &env)
  Convenience function to plug a library client’s interface into the appropriate model, demonstrating use of Dakota parallel configuration in constructing the plugin Interface on the right MPI Comm.
- static void callback_function (Dakota::ProblemDescDB *db, void *ptr)
  Example: user-provided post-parse callback (Dakota::DbCallbackFunction)
- int main (int argc, char *argv[ ])  
  A mock simulator main for testing Dakota in library mode.

Variables
- static const char serial_input [ ]
  Default Dakota input string for serial case (rosenbrock):
- static const char parallel_input [ ]
  Default Dakota input string for parallel case (text book)

14.8.1 Detailed Description
file containing a mock simulator main for testing Dakota in library mode
14.8.2 Function Documentation

**void fpinit_ASLO ( )**

Floating-point initialization from AMPL; switch to 53-bit rounding if appropriate, to eliminate some cross-platform differences.

Referenced by main().

**void run_dakota_parse ( const char * dakota_input_file )**

Run a Dakota LibraryEnvironment, mode 1: parsing an input file.

Simplest library case: this function parses from an input file to define the ProblemDescDB data.

References Environment::execute(), ProgramOptions::input_file(), Environment::mpi_manager(), MPIManager::mpirun_flag(), parallel_interface_plugin(), serial_interface_plugin(), and MPIManager::world_rank().

Referenced by main().

**void run_dakota_data ( )**

Run a Dakota LibraryEnvironment, mode 2: from C++ API inserted data.

Rather than parsing from an input file, this function populates Data class objects directly using a minimal specification and relies on constructor defaults and post-processing in post_process() to fill in the rest.

References DataInterface::data_rep(), DataResponses::data_rep(), DataVariables::data_rep(), DataMethod::data_rep(), LibraryEnvironment::done_modifying_db(), Environment::execute(), Environment::exit_mode(), DataResponsesRep::gradientType, DataResponsesRep::hessianType, LibraryEnvironment::insert_nodes(), DataMethodRep::method-Name, Environment::mpi_manager(), MPIManager::mpirun_flag(), ParallelLibrary::mpirun_flag(), DataVariables-Rep::numContinuousDesVars, DataResponsesRep::numNonlinearIneqConstraints, DataResponsesRep::numObjective-Functions, parallel_interface_plugin(), Environment::parallel_library(), serial_interface_plugin(), and ParallelLibrary::world_rank().

Referenced by main().

**void run_dakota_mixed ( const char * dakota_input_file, bool mpirun_flag )**

Run a Dakota LibraryEnvironment, from string or input file input, supplemented with additional C++ API adjustments.

Function to encapsulate the Dakota object instantiations for mode 3: mixed parsing and direct updating.

This function showcases multiple features. For parsing, either an input file (dakota_input_file != NULL) or a default input string (dakota_input_file == NULL) are shown. This parsed input is then mixed with input from three sources: (1) input from a user-supplied callback function, (2) updates to the DB prior to Environment instantiation, (3) updates directly to Iterators/Models following Environment instantiation.

References callback_function(), LibraryEnvironment::done_modifying_db(), ProgramOptions::echo_input(), Environment-::execute(), LibraryEnvironment::filtered_model_list(), ProblemDescDB::get_sa(), ProgramOptions::input_file(), ProgramOptions::input_string(), Environment::mpi_manager(), MPIManager::mpirun_flag(), parallel_input, parallel- _interface_plugin(), Environment::parallel_library(), Environment::problem_description_db(), ProblemDescDB::resolve- _top_method(), callback_data::rosen_cdv_upper_bd, serial_input, serial_interface_plugin(), ProblemDescDB::set(), and ParallelLibrary::world_rank().

Referenced by main().

**void serial_interface_plugin ( Dakota::LibraryEnvironment & env )**

Convenience function with simplest example of interface plugin: plugin a serial DirectApplicInterface that can be constructed independent of Dakota’s configuration details.
Demonstration of simple plugin where client code doesn’t require access to detailed Dakota data (such as Model-based parallel configuration information) to construct the DirectApplicInterface. This example plugs-in a derived serial direct application interface instance (“plugin_rosenbrock”).

References Dakota::abort_handler(), LibraryEnvironment::plugin_interface(), and Environment::problem_description_db().

Referenced by run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

```c
void parallel_interface_plugin ( Dakota::LibraryEnvironment & env )
```

Convenience function to plug a library client’s interface into the appropriate model, demonstrating use of Dakota parallel configuration in constructing the plugin Interface on the right MPI Comm.

From a filtered list of Model candidates, plug-in a derived direct application interface instance (“plugin_textbook” for parallel). This approach provides more complete access to the Model, e.g., for access to analysis communicators.

References Dakota::abort_handler(), Interface::assign_rep(), LibraryEnvironment::filtered_model_list(), ProblemDescDB::get_db_model_node(), Environment::problem_description_db(), and ProblemDescDB::set_db_model_nodes().

Referenced by run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

```c
static void callback_function ( Dakota::ProblemDescDB * db, void * ptr ) [static]
```

Example: user-provided post-parse callback (Dakota::DbCallbackFunction)

Example of user-provided callback function (an instance of Dakota::DbCallbackFunction) to override input provided by parsed Dakota input file or input string data.

References Dakota::contains(), ProblemDescDB::get_sa(), ProblemDescDB::get_ushort(), ProblemDescDB::resolve_top_method(), callback_data::rosen_cdv_upper_bd, and ProblemDescDB::set().

Referenced by run_dakota_mixed().

```c
int main ( int argc, char * argv[] )
```

A mock simulator main for testing Dakota in library mode.

Overall Usage: dakota_library_mode [-mixed] [dakota.in]

Uses alternative instantiation syntax as described in the library mode documentation within the Developers Manual. Tests several problem specification modes:

1. run_dakota_parse: reads all problem specification data from a Dakota input file. Usage: dakota_library_mode dakota.in
2. run_dakota_data: creates all problem specification from direct Data instance instantiations in the C++ code. Usage: dakota_library_mode
3. run_dakota_mixed: a mixture of input parsing and direct data updates, where the data updates occur: (a) via the DB during Environment instantiation, and (b) via Iterators/Models following Environment instantiation. Usage: dakota_library_mode -mixed (input from default string) dakota_library_mode -mixed dakota.in (input from specified file)

Serial cases use a plugin rosenbrock model, while parallel cases use textbook.

References MPIManager::detect_parallel_launch(), fpinit_ASL(), Dakota::mpi_debug_hold(), run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

### 14.8.3 Variable Documentation

```c
const char serial_input[] [static]
```

Initial value:
Default Dakota input string for serial case (rosenbrock):
Referenced by run_dakota_mixed().

const char parallel_input[ ] [static]

Initial value:

Default Dakota input string for parallel case (text_book)
Referenced by run_dakota_mixed().

14.9 library_split.cpp File Reference

file containing a mock simulator main for testing DAKOTA in library mode on a split communicator

Functions

- void manage_mpi (MPI_Comm &my_comm, int &color)
  
  Split MPI_COMM_WORLD, returning the comm and color.
- void gen_dakota_input (const int &color, std::string &input)
  
  Return the appropriate DAKOTA input based on color (1 or 2)
- void run_dakota (const MPI_Comm &comm, const std::string &input, const int &color)
  
  Launch DAKOTA on passed communicator, tagging output/error with color.
- void collect_results ()
  
  Wait for and collect results from DAKOTA runs.
- int main (int argc, char *argv[])
  
  Driver routine for testing library mode with partitioned MPI_Comm. This test fixture requires MPI and can be run on 3–8 processors.
14.9.1 Detailed Description
file containing a mock simulator main for testing DAKOTA in library mode on a split communicator

14.10 main.cpp File Reference
file containing the main program for DAKOTA

Functions
- void fpinit_ASLS()
- int main(int argc, char *argv[])

The main DAKOTA program.

14.10.1 Detailed Description
file containing the main program for DAKOTA

14.10.2 Function Documentation
void fpinit_ASLS()

Floating-point initialization from AMPL: switch to 53-bit rounding if appropriate, to eliminate some cross-platform differences.

int main(int argc, char *argv[])
The main DAKOTA program.

Manage command line inputs, input files, restart file(s), output streams, and top level parallel iterator communicators. Instantiate the ExecutableEnvironment and invoke its execute() virtual function.

References Environment::check(), ExecutableEnvironment::execute(), fpinit_ASLS(), Dakota::mpi_debug_hold(), and Dakota::register_signal_handlers().

14.11 restart_util.cpp File Reference
file containing the DAKOTA restart utility main program

Namespaces
- Dakota

The primary namespace for DAKOTA.

Constant Groups
- Dakota

The primary namespace for DAKOTA.
14.11. RESTART_UTIL.CPP FILE REFERENCE

Functions

- void **print_usage**(std::ostream &s)
  
  print restart utility help message

- void **print_restart**(StringArray pos_args, String print_dest)
  
  print a restart file

- void **print_restart_pdb**(StringArray pos_args, String print_dest)
  
  print a restart file (PDB format)

- void **print_restart_tabular**(StringArray pos_args, String print_dest, unsigned short tabular_format, int tabular_precision)
  
  print a restart file (tabular format)

- void **read_neutral**(StringArray pos_args)
  
  read a restart file (neutral file format)

- void **repair_restart**(StringArray pos_args, String identifier_type)
  
  repair a restart file by removing corrupted evaluations

- void **concatenate_restart**(StringArray pos_args)
  
  concatenate multiple restart files

- int **main**(int argc, char *argv[])
  
  The main program for the DAKOTA restart utility.

14.11.1 Detailed Description

file containing the DAKOTA restart utility main program

14.11.2 Function Documentation

int main ( int argc, char * argv[] )

The main program for the DAKOTA restart utility.

Parse command line inputs and invoke the appropriate utility function (**print_restart()**, **print_restart_tabular()**, **read_neutral()**, **repair_restart()**, or **concatenate_restart()**).

References Dakota::concatenate_restart(), Dakota::print_restart(), Dakota::print_restart_pdb(), Dakota::print_restart_tabular(), Dakota::print_usage(), Dakota::read_neutral(), Dakota::repair_restart(), and Dakota::write_precision.
Bibliography

## Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>~Approximation</td>
<td>207</td>
</tr>
<tr>
<td>Dakota::Approximation</td>
<td></td>
</tr>
<tr>
<td>~Constraints</td>
<td>246</td>
</tr>
<tr>
<td>Dakota::Constraints</td>
<td></td>
</tr>
<tr>
<td>~DataFitSurrModel</td>
<td>258</td>
</tr>
<tr>
<td>Dakota::DataFitSurrModel</td>
<td></td>
</tr>
<tr>
<td>~EffGlobalMinimizer</td>
<td>335</td>
</tr>
<tr>
<td>Dakota::EffGlobalMinimizer</td>
<td></td>
</tr>
<tr>
<td>~Environment</td>
<td>341</td>
</tr>
<tr>
<td>Dakota::Environment</td>
<td></td>
</tr>
<tr>
<td>~Interface</td>
<td>393</td>
</tr>
<tr>
<td>Dakota::Interface</td>
<td></td>
</tr>
<tr>
<td>~Iterator</td>
<td>403</td>
</tr>
<tr>
<td>Dakota::Iterator</td>
<td></td>
</tr>
<tr>
<td>~Model</td>
<td>468</td>
</tr>
<tr>
<td>Dakota::Model</td>
<td></td>
</tr>
<tr>
<td>~NonDAdaptiveSampling</td>
<td>526</td>
</tr>
<tr>
<td>Dakota::NonDAdaptiveSampling</td>
<td></td>
</tr>
<tr>
<td>~ProblemDescDB</td>
<td>720</td>
</tr>
<tr>
<td>Dakota::ProblemDescDB</td>
<td></td>
</tr>
<tr>
<td>~SharedApproxData</td>
<td>802</td>
</tr>
<tr>
<td>Dakota::SharedApproxData</td>
<td></td>
</tr>
<tr>
<td>_initPts</td>
<td></td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer</td>
<td>421</td>
</tr>
<tr>
<td>_model</td>
<td></td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer::Evaluator</td>
<td>349</td>
</tr>
<tr>
<td>APPSEvalMgr</td>
<td>216</td>
</tr>
<tr>
<td>Dakota::APPSValMgr</td>
<td></td>
</tr>
<tr>
<td>abmInstance</td>
<td>217</td>
</tr>
<tr>
<td>Dakota::AdaptedBasisModel</td>
<td>178</td>
</tr>
<tr>
<td>abort_handler_t</td>
<td>148</td>
</tr>
<tr>
<td>Dakota</td>
<td></td>
</tr>
<tr>
<td>abort_mode</td>
<td>156</td>
</tr>
<tr>
<td>Dakota</td>
<td></td>
</tr>
<tr>
<td>abort_throw_or_exit</td>
<td>147</td>
</tr>
<tr>
<td>Dakota</td>
<td></td>
</tr>
<tr>
<td>acceptanceChain</td>
<td>536</td>
</tr>
<tr>
<td>Dakota::NonDBayesCalibration</td>
<td></td>
</tr>
<tr>
<td>accepts_multiple_points</td>
<td>420</td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer</td>
<td></td>
</tr>
<tr>
<td>active_set_mapping</td>
<td>640</td>
</tr>
<tr>
<td>Dakota::NonDSampling</td>
<td></td>
</tr>
<tr>
<td>active_variables</td>
<td>475</td>
</tr>
<tr>
<td>Dakota::Model</td>
<td></td>
</tr>
<tr>
<td>ActiveSet</td>
<td>165</td>
</tr>
<tr>
<td>ActiveSubspaceModel</td>
<td>172</td>
</tr>
<tr>
<td>Dakota::ActiveSubspaceModel</td>
<td></td>
</tr>
<tr>
<td>activeVarsCompsTotals</td>
<td>822</td>
</tr>
<tr>
<td>Dakota::SharedVariablesDataRep</td>
<td></td>
</tr>
<tr>
<td>actualModel</td>
<td>266</td>
</tr>
<tr>
<td>Dakota::DataFitSurrModel</td>
<td></td>
</tr>
<tr>
<td>actualReducedRank</td>
<td>748</td>
</tr>
<tr>
<td>Dakota::RandomFieldModel</td>
<td></td>
</tr>
<tr>
<td>add_anchor_to_surfdata</td>
<td>838</td>
</tr>
<tr>
<td>Dakota::SurfpackApproximation</td>
<td></td>
</tr>
<tr>
<td>add_datapoint</td>
<td>376</td>
</tr>
<tr>
<td>Dakota::Graphics</td>
<td></td>
</tr>
<tr>
<td>place</td>
<td>377</td>
</tr>
<tr>
<td>Dakota::OutputManager</td>
<td>670</td>
</tr>
<tr>
<td>aggregate_acceptance_chain</td>
<td>625</td>
</tr>
<tr>
<td>Dakota::NonDQUESOBayesCalibration</td>
<td></td>
</tr>
<tr>
<td>algorithm_space_model</td>
<td>185</td>
</tr>
<tr>
<td>Dakota::Analyzer</td>
<td></td>
</tr>
<tr>
<td>place</td>
<td>179</td>
</tr>
<tr>
<td>anisotropic_order_to_dimension_preference</td>
<td>569</td>
</tr>
<tr>
<td>Dakota::NonDIIntegration</td>
<td></td>
</tr>
<tr>
<td>append_approximation</td>
<td>186</td>
</tr>
<tr>
<td>Dakota::ApproximationInterface</td>
<td></td>
</tr>
<tr>
<td>ApplicationInterface</td>
<td>261</td>
</tr>
<tr>
<td>Dakota::DataFitSurrModel</td>
<td></td>
</tr>
<tr>
<td>approx_subprob_constraint_eval</td>
<td>262</td>
</tr>
<tr>
<td>Dakota::SurrBasedLocalMinimizer</td>
<td>846</td>
</tr>
</tbody>
</table>
Dakota::NIDRProblemDescDB, 500
check_input
Dakota::ProblemDescDB, 721
check_variables
Dakota::NonDIntegration, 569
check_wait
Dakota::ProcessHandleApplicInterface, 729
clear
Dakota::ParallelLevel, 677
clear_all
Dakota::Approximation, 209
clear_current
Dakota::Approximation, 209
Dakota::TANA3Approximation, 862
Clone
Dakota::JEGAOptimizer::Evaluator, 348
colin_request_to_dakota_request
Dakota::COLINApplication, 227
CollabHybridMetaIterator, 229
collect_evaluation_impl
Dakota::COLINApplication, 227
CommandLineHandler, 231
CommandShell, 232
compute
Dakota::DiscrepancyCorrection, 329
compute_correlations
Dakota::SensAnalysisGlobal, 792
compute_covariance_metric
Dakota::NonDExpansion, 549
Dakota::NonDStochCollocation, 648
compute_densities
Dakota::NonD, 516
compute_final_statistics_metric
Dakota::NonDExpansion, 549
Dakota::NonDStochCollocation, 648
compute_level_mappings
Dakota::NonDSampling, 638
compute_statistics
Dakota::NonDExpansion, 549
compute_trust_region_ratio
Dakota::SurrBasedLocalMinimizer, 845
compute_vbd_stats
Dakota::Analyzer, 185
concatenate_restart
Dakota, 156
ConcurrentMetaIterator, 233
configure
Dakota::IteratorScheduler, 413, 414
configure_constraint_maps
Dakota::Optimizer, 666
configure_equali... Dakoda::Optimizer, 666
configure_inequality_constraint_maps
Dakoda, 150
ConsoleRedirector, 237
constraint_violation
Dakota::SurrBasedMinimizer, 851
Constraints, 239
Dakota::Constraints, 246
construct_map_optimizer
Dakota::NonDBayesCalibration, 535
contains
Dakota, 156
color_variate_mc
Dakota::NonDMultilevelSampling, 605
converge_order
Dakota::RichExtrapVerification, 781
converge_qoi
Dakota::RichExtrapVerification, 781
copy
Dakota::Constraints, 247
Dakota::SharedResponseData, 809
Dakota::SharedVariablesData, 817
copy_field_data
Dakota, 152
copy_items
Dakota::WorkdirHelper, 897
copy_rep
Dakota::SharedResponseDataRep, 810
Dakota::SharedVariablesDataRep, 822
core_run
Dakota::DDACEDesignCompExp, 320
Dakota::EffGlobalMinimizer, 335
Dakota::FSUDesignCompExp, 365
Dakota::Iterator, 405
Dakota::JEGAOptimizer, 420
Dakota::NonDAdaptiveSampling, 526
Dakota::NonDBayesCalibration, 534
Dakota::NonDGloba... Dakoda::SurrBasedMinimizer, 851
Dakota::NonDBayesCalibration, 535
Dakota::NonDMultilevelSampling, 605
Dakota::NonDLHSSampling, 577
Dakota::NonDMultilevelSampling, 603
Dakota::NonDRKDDarts, 631
Dakota::NonDSampling, 639
Dakota::NonlinearCGOptimizer, 654
Dakota::NOWPACOptimizer, 657
Dakota::ParamStudy, 694
set_compare, 154
set_variables, 150
start_dakota_heartbeat, 153
start_grid_computing, 153
stop_grid_computing, 153
dakota::GetLongOpt
Dakota::GetLongOpt
MandatoryValue, 374
OptionalValue, 374
Valueless, 374
Dakota::APPSEvalMgr
APPSEvalMgr, 217
isReadyForWork, 217
recv, 217
submit, 217
Dakota::ActiveSet
derivVarsVector, 167
requestVector, 167
Dakota::ActiveSubspaceModel
ActiveSubspaceModel, 172
asmInstance, 174
build_cv_surrogate, 173
build_surrogate, 173
derived_init_communicators, 172
initialize_mapping, 172
initialize_recast, 173
response_mapping, 174
set_mapping, 174
uncertain_vars_to_subspace, 173
vars_mapping, 173
Dakota::AdaptedBasisModel
abmInstance, 178
derived_init_communicators, 177
initialize_recast, 177
response_mapping, 178
set_mapping, 178
uncertain_vars_to_subspace, 177
vars_mapping, 178
Dakota::Analyzer
algorithm_space_model, 185
compute_vbd_stats, 185
evaluate_parameter_sets, 185
finalize_run, 184
get_vbd_parameter_sets, 185
initialize_run, 183
num_samples, 182
post_run, 183
pre_output, 184
pre_run, 183
print_results, 184
print_sobol_indices, 186
read_variables_responses, 185
sample_to_variables, 183
variables_to_sample, 186
Dakota::ApplicationInterface
asynchronous_local_evaluations, 198
asynchronous_local_evaluations_nowait, 200
duplication_detect, 197
init_communicators_checks, 196
init_default_asv, 197
init_serial, 194
map, 194
master_dynamic_schedule_analyses, 196
master_dynamic_schedule_evaluations, 197
master_dynamic_schedule_evaluations_nowait, 199
peer_dynamic_schedule_evaluations, 198
peer_dynamic_schedule_evaluations_nowait, 200
peer_static_schedule_evaluations, 198
peer_static_schedule_evaluations_nowait, 199
serve_analyses_synch, 196
serve_evaluations, 195
serve_evaluations_asynch, 201
serve_evaluations_asynch_peer, 201
serve_evaluations_synch, 201
serve_evaluations_synch_peer, 201
set_communicators_checks, 196
stop_evaluation_servers, 195
synchronize, 195
synchronize_nowait, 195
synchronous_local_evaluations, 199
Dakota::Approximation
symmetric_eigenvalue_decomposition, 152
VLUncertainInt, 159
VLUncertainReal, 159
VLUncertainStr, 160
var_mp_cbound, 162
var_mp_check_cau, 160
var_mp_check_ceu, 161
var_mp_check_cv, 160
var_mp_check_daui, 161
var_mp_check_daur, 161
var_mp_check_daus, 161
var_mp_check_deui, 161
var_mp_check_deur, 162
var_mp_check_deus, 161
var_mp_check_dset, 160
var_mp_drange, 162
write_ordered, 151, 152
Dakota::ActiveSubspaceModel
dervariables, 150
Dakota::ApplicationInterface
asynchronous_local_evaluations, 198
asynchronous_local_evaluations_nowait, 200
duplication_detect, 197
init_communicators_checks, 196
init_default_asv, 197
init_serial, 194
map, 194
master_dynamic_schedule_analyses, 196
master_dynamic_schedule_evaluations, 197
master_dynamic_schedule_evaluations_nowait, 199
peer_dynamic_schedule_evaluations, 198
peer_dynamic_schedule_evaluations_nowait, 200
peer_static_schedule_evaluations, 198
peer_static_schedule_evaluations_nowait, 199
serve_analyses_synch, 196
serve_evaluations, 195
serve_evaluations_asynch, 201
serve_evaluations_asynch_peer, 201
serve_evaluations_synch, 201
serve_evaluations_synch_peer, 201
set_communicators_checks, 196
stop_evaluation_servers, 195
synchronize, 195
synchronize_nowait, 195
synchronous_local_evaluations, 199
Dakota::Approximation
~Approximation, 207
Approximation, 206, 207
build, 207
clear_all, 209
clear_current, 209
export_model, 208
finalize, 208
get_approx, 209
operator=, 207
pop, 208
push, 208
rebuild, 208
Dakota::ApproximationInterface
append_approximation, 213, 214
build_approximation, 214
export_approximation, 214
functionSurfaces, 215
pop_approximation, 214
push_approximation, 215
read_challenge_points, 215
rebuild_approximation, 214
update_approximation, 213
Dakota::BootstrapSampler
operator(), 221
Dakota::BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType >>
operator(), 222
Dakota::BootstrapSamplerBase
bootstrapRNG, 224
Dakota::COLINApplication
colin_request_to_dakota_request, 227
collect_evaluation_impl, 227
dakota_response_to_colin_response, 228
evaluation_available, 227
map_domain, 228
perform_evaluation_impl, 227
set_problem, 226
spawn_evaluation_impl, 227
Dakota::CommandLineHandler
output_helper, 232
Dakota::CommandShell
flush, 233
operator<<, 233
Dakota::ConcurrentMetaIterator
pre_run, 236
print_results, 236
Dakota::Constraints
~Constraints, 246
Constraints, 246
copy, 247
get_constraints, 248
manage_linear_constraints, 247
operator=, 247
reshape, 247
shape, 247
Dakota::DDACEDesignCompExp
core_run, 320
DDACEDesignCompExp, 319
num_samples, 320
post_run, 320
pre_run, 320
resolve_samples_symbols, 321
Dakota::DataFitSurrModel
~DataFitSurrModel, 258
actualModel, 266
append_approximation, 261, 262
build_approximation, 260
build_global, 263
build_local_multipoint, 264
derived_evaluate, 258
derived_evaluate_nowait, 259
derived_init_communicators, 262
derived_synchronize, 259
derived_synchronize_nowait, 259
export_point, 263
finalize_export, 263
finalize_mapping, 258
import_points, 263
init_model, 264
initialize_export, 263
update_approximation, 260, 261
update_from_model, 265
update_model, 265
Dakota::DataTransformModel
DataTransformModel, 299
derived_evaluate, 300
derived_evaluate_nowait, 300
derived_synchronize, 300
derived_synchronize_nowait, 301
dtModelInstance, 302
expand_scales_array, 302
filter_submodel_responses, 301
init_continuous_vars, 301
set_mapping, 301
transform_response_map, 301
variables_expand, 300
Dakota::DirectApplicInterface
derived_map_ac, 325
init_communicators_checks, 325
set_communicators_checks, 325
synchronous_local_analysis, 325
Dakota::DiscrepancyCorrection
  compute, 329
Dakota::EffGlobalMinimizer
  ~EffGlobalMinimizer, 335
  algorithm_space_model, 335
  core_run, 335
  get_best_sample, 335
Dakota::Environment
  ~Environment, 341
  Environment, 340, 341
  exit_mode, 341
  get_environment, 342
  operator=, 341
  parse, 342
Dakota::ExperimentData
  cov_determinant, 357
  form_residuals, 356
  half_log_cov_det_gradient, 357
  half_log_cov_det_hessian, 357
  half_log_cov_determinant, 357
  load_experiment, 358
  parse_sigma_types, 358
  recover_model, 356
  residuals_per_multiplier, 358
  scale_residuals, 357
Dakota::FSUDesignCompExp
  core_run, 365
  enforce_input_rules, 366
  FSUDesignCompExp, 365
  num_samples, 366
  post_run, 365
  pre_run, 365
Dakota::GaussProcApproximation
  GPmodel_apply, 371
  GaussProcApproximation, 371
trendOrder, 371
Dakota::GetLongOpt
  enroll, 374
  GetLongOpt, 374
  OptType, 374
  parse, 374
  retrieve, 375
  usage, 375
Dakota::Graphics
  add_datapoint, 376, 377
  create_plots_2d, 376
  new_dataset, 377
Dakota::GridApplicInterface
  synchronous_local_analysis, 379
Dakota::HierarchSurrModel
  derived_evaluate, 384
  derived_evaluate_nowait, 385
  derived_synchronize, 385
  derived_synchronize_nowait, 385
  finalize_mapping, 384
  initialize_mapping, 384
Dakota::Interface
  ~Interface, 393
  assign_rep, 393
  eval_tag_prefix, 394
  get_interface, 394
  Interface, 393
  operator=, 393
  rawResponseMap, 395
  response_mapping, 394
Dakota::Iterator
  ~Iterator, 403
  assign_rep, 407
  core_run, 405
  eval_tag_prefix, 407
  finalize_run, 405
  get_iterator, 408
  gnewton_set_recast, 407
  initialize_graphics, 405
  initialize_run, 404
  Iterator, 403, 404
  maxEvalConcurrency, 409
  operator=, 404
  post_run, 405
  pre_run, 404
  print_result, 406
  probDescDB, 408
  run, 406
  uses_method, 406
Dakota::IteratorScheduler
  configure, 413, 414
  free_iterator, 413
  init_iterator, 412, 413
  IteratorScheduler, 412
  master_dynamic_schedule_iterators, 415
  partition, 414
  run_iterator, 413
  schedule_iterators, 414
  serve_iterators, 415
  set_iterator, 413
Dakota::JEGAOptimizer
_InitPts, 421
accepts_multiple_points, 420
core_run, 420
GetBestMOSolutions, 419
GetBestSOSolutions, 419
GetBestSolutions, 419
_initial_points, 420, 421
JEGAOptimizer, 417
LoadAlgorithmConfig, 418
LoadDakotaResponses, 418
LoadProblemConfig, 418
LoadTheConstraints, 419
LoadTheDesignVariables, 418
LoadTheParameterDatabase, 418
returns_multiple_points, 420
ToDoubleMatrix, 420
Dakota::JEGAOptimizer::Driver
DestroyAlgorithm, 332
Driver, 332
ExtractAllData, 332
PerformIterations, 332
Dakota::JEGAOptimizer::Evaluator
_model, 349
Clone, 348
Description, 346
Evaluate, 347, 348
Evaluator, 345, 346
GetDescription, 348
GetName, 348
GetNumberLinearConstraints, 347
GetNumberNonLinearConstraints, 347
Name, 346
RecordResponses, 347
SeparateVariables, 346
Dakota::JEGAOptimizer::EvaluatorCreator
CreateEvaluator, 349
EvaluatorCreator, 349
Dakota::LabelsWriter
_operator(), 422
Dakota::LeastSq
_finalize_run, 425
get_confidence_intervals, 425
initialize_run, 424
LeastSq, 424
_post_run, 425
primary_resp_weighter, 426
print_results, 425
weight_model, 426
Dakota::LibraryEnvironment
filtered_interface_list, 428
filtered_model_list, 428
LibraryEnvironment, 427
Dakota::MatlabInterface
derived_map_ac, 430
matlab_engine_run, 430
Dakota::MetaIterator
_post_run, 432
Dakota::Minimizer
algorithm_space_model, 438
data_transform_model, 438
_finalize_run, 437
_initialize_run, 437
local_recast_retrieve, 439
Minimizer, 437
objective, 438
objective_gradient, 439
objective_hessian, 439
_post_run, 437
resize_best_resp_array, 439
resize_best_vars_array, 439
scale_model, 438
Dakota::MixedVarConstraints
MixedVarConstraints, 441
Dakota::MixedVariables
MixedVariables, 442
read_core, 442
read_tabular, 442
Dakota::Model
~Model, 468
active_variables, 475
assign_rep, 474
derivative_concurrency, 474
derived_interface, 470
estimate_derivatives, 475
estimate_message_lengths, 473
eval_tag_prefix, 472
evaluation_cache, 472
FDstep1, 477
FDstep2, 477
fdGradStepSize, 478
fdHessByFnStepSize, 478
fdHessByGradStepSize, 478
get_model, 475
inactive_variables, 475
init_communicators, 473
init_serial, 473
initialize_h, 477
interface_id, 471
local_eval_concurrency, 471
local_eval_synchronization, 471
manage_asv, 477
manage_data_recastings, 474
Model, 468
operator=, 469
probDescDB, 478
restart_file, 472
solution_level_index, 471
solution_levels, 471
subordinate_iterator, 469
subordinate_model, 469
subordinate_models, 472
surrogate_model, 469
synchronize_derivatives, 476
truth_model, 470
update_from_subordinate_model, 470
update_quasi_hessians, 476
update_response, 476
Dakota::NIDRProblemDescDB
  check_driver, 500
derived_parse_inputs, 500
make_variable_defaults, 500
Dakota::NOWPACOptimizer
core_run, 657
Dakota::NestedModel
cnv_index_map, 493
cdiv_index_map, 494
cdrv_index_map, 494
cdsv_index_map, 494
cv_index_map, 493
derived_evaluate, 491
derived_evaluate_nowait, 491
derived_evaluation_id, 492
derived_init_communicators, 492
derived_master_overload, 492
derived_synchronize, 491
div_index_map, 493
drv_index_map, 493
dsv_index_map, 493
local_eval_concurrency, 492
local_eval_synchronization, 492
response_mapping, 494
subModel, 495
Dakota::NomadOptimizer::Evaluator
eval_x, 343
Evaluator, 343
Dakota::NonD
  compute_densities, 516
  finalize_run, 515
  initialize_final_statistics, 516
  initialize_random_variable_parameters, 515
  initialize_random_variable_types, 514
  initialize_random_variables, 514
  initialize_run, 515
  level_mappings_file, 517
  pre_run, 515
  print_level_map, 517
  print_level_mappings, 515
  set_u_to_x_mapping, 517
  vars_u_to_x_mapping, 516
  vars_x_to_u_mapping, 516
Dakota::NonDAdaptImpSampling
  initialize, 521
  NonDAdaptImpSampling, 520
Dakota::NonDAdaptiveSampling
  ~NonDAdaptiveSampling, 526
core_run, 526
  NonDAdaptiveSampling, 526
  print_results, 526
Dakota::NonDBayesCalibration
  acceptanceChain, 536
  algorithm_space_model, 534
  construct_map_optimizer, 535
core_run, 534
  export_chain, 535
  log_likelihood, 535
  neg_log_post_respond_mapping, 535
  NonDBayesCalibration, 533
  print_results, 534
  prior_mean, 534
  prior_variance, 534
Dakota::NonDCalibration
  NonDCalibration, 537
Dakota::NonDCubature
  increment_grid_preference, 539
  increment_reference, 540
  NonDCubature, 539
  num_samples, 540
  sampling_reset, 539
Dakota::NonDREAMBayesCalibration
  cache_chain, 543
calibrate, 543
  NonDREAMBayesCalibration, 542
  prior_density, 542
  prior_sample, 543
problem_size, 542
problem_value, 542
sample_likelihood, 543
Dakota::NonExpansion
algorithm_space_model, 549
calculate_covariance_metric, 549
calculate_final_statistics_metric, 549
calculate_statistics, 549
useDerivs, 550
Dakota::NonDGPImpSampling
core_run, 562
NonDGPImpSampling, 561
Dakota::NonDGPMSABayesCalibration
cache_acceptance_chain, 566
calibrate, 565
fill_simulation_data, 566
NonDGPMSABayesCalibration, 565
print_results, 566
Dakota::NonDGlobalInterval
algorithm_space_model, 555
Dakota::NonDGlobalReliability
core_run, 557
print_results, 557
Dakota::NonDIntegration
anisotropic_order_to_dimension_preference, 569
check_variables, 569
core_run, 569
dimension_preference_to_anisotropic_order, 569
NonDIntegration, 568
Dakota::NonDLHSSampling
core_run, 577
d_optimal_parameter_set, 578
NonDLHSSampling, 577
Dakota::NonDLocalReliability
dg_ds_eval, 591
dp2_dbeta_factor, 592
initial_taylor_series, 589
initialize_class_data, 590
initialize_level_data, 590
initialize_mpp_search_data, 590
PMA2_constraint_eval, 589
PMA_constraint_eval, 589
PMA_objective_eval, 588
probability, 592
RIA_constraint_eval, 588
RIA_objective_eval, 588
update_level_data, 591
update_mpp_search_data, 591
Dakota::NonDMultilevelPolynomialChaos
NonDMultilevelPolynomialChaos, 596
Dakota::NonDMultilevelSampling
control_variate_mc, 605
core_run, 603
multilevel_control_variate_mc_Qcorr, 605
multilevel_control_variate_mc_Ycorr, 605
multilevel_mc_Qsum, 604
multilevel_mc_Ysum, 604
NonDMultilevelSampling, 603
post_run, 603
pre_run, 603
print_results, 604
uncentered_to_centered, 606
Dakota::NonDMultilevelStochCollocation
NonDMultilevelStochCollocation, 607, 608
Dakota::NonDPolynomialChaos
increment_grid_from_order, 616
increment_order_and_grid, 616
increment_order_from_grid, 616
NonDPolynomialChaos, 615, 616
Dakota::NonDQUESOBayesCalibration
aggregate_acceptance_chain, 625
calibrate, 625
init_parameter_domain, 626
logitTransform, 627
NonDQUESOBayesCalibration, 625
print_results, 625
prior_proposal_covariance, 626
set_ip_options, 626
user_proposal_covariance, 626
Dakota::NonDQuadrature
initialize_grid, 620
NonDQuadrature, 619, 620
num_samples, 620
sampling_reset, 620
Dakota::NonDRKDDarts
core_run, 631
post_run, 631
pre_run, 631
Dakota::NonDReliability
algorithm_space_model, 629
Dakota::NonDSampling
active_set_mapping, 640
compute_level_mappings, 638
core_run, 639
get_parameter_sets, 639, 640
mode_counts, 642
NonDSampling, 637, 638
num_samples, 639
Dakota::NonDStochasticOptimization
active_set_mapping, 640
compute_quadratic_cholesky, 638
compute_statistics, 639
create_quadratic_cholesky, 638
initialize_class_data, 639
initialize_level_data, 639
initialize_mpp_search_data, 639
PMA2_constraint_eval, 639
PMA_constraint_eval, 639
PMA_objective_eval, 639
probability, 642
RIA_constraint_eval, 642
RIA_objective_eval, 642
update_level_data, 642
update_mpp_search_data, 642
Dakota::NonDMultilevelSampling
control_variate_mc, 605
core_run, 603
multilevel_control_variate_mc_Qcorr, 605
multilevel_control_variate_mc_Ycorr, 605
multilevel_mc_Qsum, 604
multilevel_mc_Ysum, 604
NonDMultilevelSampling, 603
post_run, 603
pre_run, 603
print_results, 604
uncentered_to_centered, 606
Dakota::NonDMultilevelStochCollocation
NonDMultilevelStochCollocation, 607, 608
Dakota::NonDPolynomialChaos
increment_grid_from_order, 616
increment_order_and_grid, 616
increment_order_from_grid, 616
NonDPolynomialChaos, 615, 616
Dakota::NonDQUESOBayesCalibration
aggregate_acceptance_chain, 625
calibrate, 625
init_parameter_domain, 626
logitTransform, 627
NonDQUESOBayesCalibration, 625
print_results, 625
prior_proposal_covariance, 626
set_ip_options, 626
user_proposal_covariance, 626
Dakota::NonDQuadrature
initialize_grid, 620
NonDQuadrature, 619, 620
num_samples, 620
sampling_reset, 620
Dakota::NonDRKDDarts
core_run, 631
post_run, 631
pre_run, 631
Dakota::NonDReliability
algorithm_space_model, 629
Dakota::NonDSampling
active_set_mapping, 640
compute_quadratic_cholesky, 638
compute_statistics, 639
create_quadratic_cholesky, 638
initialize_class_data, 639
initialize_level_data, 639
initialize_mpp_search_data, 639
PMA2_constraint_eval, 639
PMA_constraint_eval, 639
PMA_objective_eval, 639
probability, 642
RIA_constraint_eval, 642
RIA_objective_eval, 642
update_level_data, 642
update_mpp_search_data, 642
Dakota::NonDMultilevelPolynomialChaos
NonDMultilevelPolynomialChaos, 596
Dakota::NonDMultilevelSampling
control_variate_mc, 605
core_run, 603
multilevel_control_variate_mc_Qcorr, 605
multilevel_control_variate_mc_Ycorr, 605
multilevel_mc_Qsum, 604
multilevel_mc_Ysum, 604
NonDMultilevelSampling, 603
post_run, 603
pre_run, 603
print_results, 604
uncentered_to_centered, 606
Dakota::NonDMultilevelStochCollocation
NonDMultilevelStochCollocation, 607, 608
Dakota::NonDPolynomialChaos
increment_grid_from_order, 616
increment_order_and_grid, 616
increment_order_from_grid, 616
NonDPolynomialChaos, 615, 616
Dakota::NonDQUESOBayesCalibration
aggregate_acceptance_chain, 625
calibrate, 625
init_parameter_domain, 626
logitTransform, 627
NonDQUESOBayesCalibration, 625
print_results, 625
prior_proposal_covariance, 626
set_ip_options, 626
user_proposal_covariance, 626
Dakota::NonDQuadrature
initialize_grid, 620
NonDQuadrature, 619, 620
num_samples, 620
sampling_reset, 620
Dakota::NonDRKDDarts
core_run, 631
post_run, 631
pre_run, 631
Dakota::NonDReliability
algorithm_space_model, 629
Dakota::NonDSampling
active_set_mapping, 640
compute_quadratic_cholesky, 638
compute_statistics, 639
create_quadratic_cholesky, 638
initialize_class_data, 639
initialize_level_data, 639
initialize_mpp_search_data, 639
PMA2_constraint_eval, 639
PMA_constraint_eval, 639
PMA_objective_eval, 639
probability, 642
RIA_constraint_eval, 642
RIA_objective_eval, 642
update_level_data, 642
update_mpp_search_data, 642
pre_run, 639
sampling_reset, 639
transform_samples, 639
variables_to_sample, 640
view_aleatory_uncertain_counts, 641
view_design_counts, 641
view_epistemic_uncertain_counts, 641
view_uncertain_counts, 641
Dakota::NonDSparseGrid
NonDSparseGrid, 644
num_samples, 645
sampling_reset, 645
Dakota::NonDStochCollocation
compute_covariance_metric, 648
calculate_further_statistics_metric, 648
NonDStochCollocation, 647
Dakota::NonDWASABIBayesCalibration
calibrate, 651
NonDWASABIBayesCalibration, 651
print_results, 651
Dakota::NonlinearCGOptimizer
brent_minimize, 654
core_run, 654
Dakota::Optimizer
core_run, 666
load_distribute_points, 665
print_results, 665
reduce_model, 666
Dakota::OutputManager
add_datapoint, 670
create_tabular_datastream, 670
OutputManager, 670
pop_output_tag, 670
Dakota::PSUADEDesignCompExp
core_run, 739
enforce_input_rules, 740
num_samples, 740
PSUADEDesignCompExp, 739
post_run, 739
pre_run, 739
Dakota::PStudyDACE
print_results, 736
volumetric_quality, 736
Dakota::ParallelConfiguration
mi_parallel_level, 673
Dakota::ParallelLevel
clear, 677
Dakota::ParallelLibrary
compute_covariance_metric, 686
calculate_final_statistics, 686
ParallelLibrary, 685
post_run, 685
Dakota::ParamResponsePair
evalInterfaceIds, 690
ParamResponsePair, 689
write, 689
Dakota::ParamStudy
build, 694
distribute_list_of_points, 695
load_distribute_points, 695
post_run, 695
pre_run, 694
Dakota::PecosApproximation
build, 703
finalization_points, 704
pop, 704
push, 704
rebuild, 703
Dakota::ProbabilityTransformModel
initialize_random_variable_parameters, 710
initialize_random_variable_types, 709, 710
initialize_random_variables, 709
ptmInstance, 711
set_u_to_x_mapping, 711
vars_u_to_x_mapping, 710
vars_x_to_u_mapping, 710
Dakota::ProblemDescDB
~ProblemDescDB, 720
check_and_broadcast, 721
check_input, 721
generate_output, 721
operator=, 720
parse_inputs, 720
post_process, 721
ProblemDescDB, 720
Dakota::ProcessApplicInterface
autotag_files, 725
file_cleanup, 725
prepare_process_environment, 725
read_results_file, 726
reset_process_environment, 726
synchronous_local_analyses, 725
Dakota::ProcessHandleApplicInterface
asynchronous_local_analyses, 729
check_wait, 729
create_command_arguments, 730
create_evaluation_process, 728
init_communicators_checks, 728
ProcessHandleApplicInterface, 728
serve_analyses_async, 729
set_communicators_checks, 728
synchronous_local_analysis, 728
Dakota::ProgramOptions
split_filenames, 735
Dakota::PythonInterface
derived_map_ac, 742
python_convert_int, 742
Dakota::QuesoJointPdf
QuesoJointPdf, 743
Dakota::QuesoVectorRV
QuesoVectorRV, 743
Dakota::RandomFieldModel
actualReducedRank, 748
daceIterator, 748
expansionForm, 748
gen_field_data, 746
initialize_mapping, 746
initialize_recast, 747
initialize_rf_coeffs, 747
mappingInitialized, 748
rf_suite_identify_field_model, 747
rfmInstance, 748
variables_resize, 747
vars_mapping, 747
Dakota::RecastModel
derived_evaluate, 757
eval_tag_prefix, 757
init_maps, 757
RecastModel, 756
update_from_model, 757
Dakota::RelaxedVarConstraints
RelaxedVarConstraints, 760
Dakota::RelaxedVariables
read_core, 762
read_tabular, 762
RelaxedVariables, 762
Dakota::Response
function_gradients, 770
Dakota::ResultsDBAny
array_insert, 773
extract_data, 773
insert, 773
Dakota::ResultsEntry
ResultsEntry, 774
Dakota::RichExtrapVerification
converge_order, 781
converge_qoi, 781
core_run, 781
estimate_order, 781
print_results, 781
Dakota::ScalingModel
cv_scaled2native, 785
initialize_scaling, 786
lin_coeffs_modify_n2s, 786
modify_n2s, 787
modify_s2n, 787
need_resnp_trans_byvars, 787
resp_scaled2native, 785
response_modify_n2s, 788
response_modify_s2n, 786
scaleModelInstance, 788
ScalingModel, 785
secondary_resp_scaled2native, 785
secondary_resp_scaler, 787
variables_scaler, 787
Dakota::SensAnalysisGlobal
compute_correlations, 792
partial_corr, 793
simple_corr, 792
values_to_ranks, 792
Dakota::SeqHybridMetaIterator
extract_parameter_sets, 796
print_results, 796
run_sequential, 796
run_sequential_adaptive, 796
Dakota::SharedApproxData
~SharedApproxData, 802
buildDataOrder, 803
get_shared_data, 803
operator=, 803
SharedApproxData, 802
Dakota::SharedResponseData
copy, 809
Dakota::SharedResponseDataRep
copy_rep, 810
Dakota::SharedSurfpackApproxData
  SharedSurfpackApproxData, 812
Dakota::SharedVariablesData
copy, 817
Dakota::SharedVariablesDataRep
  activeVarsCompsTotals, 822
  allContinuousIds, 822
  copy_rep, 822
  inactiveVarsCompsTotals, 822
  variablesCompsTotals, 822
Dakota::SimulationModel
eval_tag_prefix, 825
Dakota::SurfpackApproximation
  add_anchor_to_surfdata, 838
  build, 837
  hessian, 837
  SurfpackApproximation, 836
  surrogates_to_surf_data, 837
Dakota::SurrogateModel
  approxBuilds, 855
  derived_evaluation_id, 854
  force_rebuild, 855
  responseMode, 855
Dakota::SysCallApplicInterface
  init_communicators_checks, 858
  set_communicators_checks, 858
  spawn_analysis_to_shell, 859
  spawn_evaluation_to_shell, 858
  spawn_input_filter_to_shell, 858
  spawn_output_filter_to_shell, 859
  synchronous_local_analysis, 858
  test_local_evaluations, 857
  wait_local_evaluations, 857
Dakota::TANA3Approximation
  build, 862
  clear_current, 862
Dakota::TabularReader
gooperator(), 860
Dakota::TabularWriter
gooperator(), 861
Dakota::TaylorApproximation
  build, 864
Dakota::TestDriverInterface
  barnes, 869
  barnes_lf, 869
  derived_map_ac, 868
  herbie, 870
  herbie1D, 869
  levenshtein_distance, 870
  lf_poly_prod, 868
  mc_api_run, 870
  poly_prod, 868
  separable_combine, 870
  shubert1D, 870
  smooth_herbie, 870
  smooth_herbie1D, 869
  steady_state_diffusion_1d, 868
  steel_column_cost, 869
Dakota::TrackerHTTP
  send_data_using_get, 873
  send_data_using_post, 873
Dakota::UsageTracker
  UsageTracker, 876
Dakota::VPSApproximation
  VPSinstance, 893
  VPSmodel_apply, 893
Dakota::Variables
  discrete_string_variables_view, 887
Dakota::Verification
  print_results, 888
Dakota::WorkdirHelper
  copy_items, 897
  create_directory, 897
  file_op_items, 898
  init_preferred_env_path, 899
  init_startup_path, 899
  initialize, 896
  link, 897
link_items, 897
po_which, 898
prepend_path_item, 898
prepend_preferred_env_path, 896
recursive_copy, 898
set_preferred_path, 898
split_wildcard, 897
tokenize_env_path, 899
which, 896
dakota_dll_api.cpp, 901
dakota_stop, 902
dakota_dll_api.h, 902
dakota_stop, 903
dakota_linear_algebra.hpp, 903
dakota_response_to_colin_response
  Dakota::COLINApplication, 228
dakota_stop
  dakota_dll_api.cpp, 902
dakota_dll_api.h, 903
dakota_tabular_io.hpp, 904
data_transform_model
  Dakota::Minimizer, 438
DataEnvironment, 248
DataEnvironmentRep, 249
DataFitSurrBasedLocalTraits, 251
DataFitSurrModel, 252
DataInterface, 266
DataMethod, 267
DataMethodRep, 268
DataModel, 284
DataModelRep, 285
DataResponses, 291
DataResponsesRep, 293
DataTransformModel, 297
  Dakota::DataTransformModel, 299
DataVariables, 302
DataVariablesRep, 304
derivVarsVector
  Dakota::ActiveSet, 167
derivative_concurrency
  Dakota::Model, 474
derived_evaluate
  Dakota::DataFitSurrModel, 258
  Dakota::DataTransformModel, 300
  Dakota::HierarchSurrModel, 384
  Dakota::NestedModel, 491
  Dakota::RecastModel, 757
derived_evaluate_nowait
  Dakota::DataFitSurrModel, 259
Dakota::DataTransformModel, 300
Dakota::HierarchSurrModel, 385
Dakota::NestedModel, 491
derived_evaluation_id
Dakota::NestedModel, 492
Dakota::SurrogateModel, 854
derived_init_communicators
Dakota::ActiveSubspaceModel, 172
Dakota::AdaptedBasisModel, 177
Dakota::DataFitSurrModel, 262
Dakota::NestedModel, 492
derived_interface
Dakota::Model, 470
derived_map_ac
Dakota::DirectApplicInterface, 325
Dakota::MatlabInterface, 430
Dakota::PythonInterface, 742
Dakota::TestDriverInterface, 868
derived_master_overload
Dakota::NestedModel, 492
derived_parse_inputs
Dakota::NIDRProblemDescDB, 500
derived_synchronize
Dakota::DataFitSurrModel, 259
Dakota::DataTransformModel, 300
Dakota::HierarchSurrModel, 385
Dakota::NestedModel, 491
derived_synchronize_nowait
Dakota::DataFitSurrModel, 259
Dakota::DataTransformModel, 301
Dakota::HierarchSurrModel, 385
Description
Dakota::JEGAOptimizer::Evaluator, 346
DesignAndStateLabelsCheck
Dakota, 159
DestroyAlgorithm
Dakota::JEGAOptimizer::Driver, 332
dg_ds_eval
Dakota::NonDLocalReliability, 591
dimension_preference_to_anisotropic_order
Dakota::NonDIntegration, 569
DirectApplicInterface, 321
DiscSetLbl
Dakota, 158
DiscrepancyCorrection, 325
discrete_string_variables_view
Dakota::Variables, 887
distribute_list_of_points
Dakota::ParamStudy, 695
Dakota::NestedModel, 493
dll_tester.cpp, 906
dp2_dbeta_factor
Dakota::NonDLocalReliability, 592
Driver
Dakota::JEGAOptimizer::Driver, 332
drv_index_map
Dakota::NestedModel, 493
dsv_index_map
Dakota::NestedModel, 493
dtModelInstance
Dakota::DataTransformModel, 302
duplication_detect
Dakota::ApplicationInterface, 197
EffGlobalMinimizer, 333
EffGlobalTraits, 335
EmbedHybridMetaIterator, 336
enforce_input_rules
Dakota::FSUDesignCompExp, 366
Dakota::PSUADesignCompExp, 740
enroll
Dakota::GetLongOpt, 374
Environment, 338
Dakota::Environment, 340, 341
estimate_derivatives
Dakota::Model, 475
estimate_message_lengths
Dakota::Model, 473
estimate_order
Dakota::RichExtrapVerification, 781
eval_tag_prefix
Dakota::Interface, 394
Dakota::Iterator, 407
Dakota::Model, 472
Dakota::RecastModel, 757
Dakota::SimulationModel, 825
eval_x
Dakota::NomadOptimizer::Evaluator, 343
evalInterfaceIds
Dakota::ParamResponsePair, 690
Evaluate
Dakota::JEGAOptimizer::Evaluator, 347, 348
evaluate_parameter_sets
Dakota::Analyzer, 185
evaluation_available
Dakota::COLINApplication, 227
evaluation_cache
Dakota::Model, 472
Evaluator
Dakota::JEGAOptimizer::Evaluator, 345, 346
Dakota::NomadOptimizer::Evaluator, 343
EvaluatorCreator
Dakota::JEGAOptimizer::EvaluatorCreator, 349
executableEnvironment, 350
exit_mode
Dakota::Environment, 341
expand_scales_array
Dakota::DataTransformModel, 302
expansionForm
Dakota::RandomFieldModel, 748
ExperimentData, 350
ExperimentResponse, 358
export_approximation
Dakota::ApproximationInterface, 214
export_chain
Dakota::NonDBayesCalibration, 535
export_model
Dakota::Approximation, 208
export_point
Dakota::DataFitSurrModel, 263
extract_data
Dakota::ResultsDBAny, 773
extract_parameter_sets
Dakota::SeqHybridMetaIterator, 796
ExtractAllData
Dakota::JEGAOptimizer::Driver, 332
FDstep1
Dakota::Model, 477
FDstep2
Dakota::Model, 477
FIELD_NAMES
Dakota, 157
FSUDesignCompExp, 362
Dakota::FSUDesignCompExp, 365
fdGradStepSize
Dakota::Model, 478
fdHessByFnStepSize
Dakota::Model, 478
fdHessByGradStepSize
Dakota::Model, 478
file_cleanup
Dakota::ProcessApplicInterface, 725
file_op_items
Dakota::WorkdirHelper, 898
FileReadException, 360
fill_simulation_data
Dakota::NonDPMSABayesCalibration, 566
INDEX

filter_submodel_responses
  Dakota::DataTransformModel, 301
filtered_interface_list
  Dakota::LibraryEnvironment, 428
filtered_model_list
  Dakota::LibraryEnvironment, 428
finalize
  Dakota::Approximation, 208
  Dakota::PecosApproximation, 704
finalize_export
  Dakota::DataFitSurrModel, 263
finalize_mapping
  Dakota::DataFitSurrModel, 258
  Dakota::HierarchSurrModel, 384
finalize_run
  Dakota::Analyzer, 184
  Dakota::Iterator, 405
  Dakota::LeastSq, 425
  Dakota::Minimizer, 437
  Dakota::NonD, 515
  Dakota::Optimizer, 665
flush
  Dakota, 147
  Dakota::CommandShell, 233
force_rebuild
  Dakota::SurrogateModel, 855
ForkApplicInterface, 360
form_residuals
  Dakota::ExperimentData, 356
fpinit_ASL
  library_mode.cpp, 909
  main.cpp, 912
free_iterator
  Dakota::IteratorScheduler, 413
FunctionEvalFailure, 366
functionGradients
  Dakota::Response, 770
functionSurfaces
  Dakota::ApproximationInterface, 215
GPmodel_apply
  Dakota::GaussProcApproximation, 371
GaussProcApproximation, 367
  Dakota::GaussProcApproximation, 371
GeneralReader, 371
GeneralWriter, 372
get_approx
  Dakota::Approximation, 209
get_best_sample
  Dakota::EffGlobalMinimizer, 335
get_bounds
  Dakota, 149
get_confidence_intervals
  Dakota::LeastSq, 425
get_constraints
  Dakota::Constraints, 248
get_db
  Dakota::ProblemDescDB, 721
get_environment
  Dakota::Environment, 342
get_field_data
  Dakota::RandomFieldModel, 746
get_interface
  Dakota::Interface, 394
get_iterator
  Dakota::Iterator, 408
get_linear_constraints
  Dakota, 150
get_mixed_bounds
  Dakota, 149
get_model
  Dakota::Model, 475
get_nonlinear_constraints
  Dakota, 151
get_nonlinear_eq_constraints
  Dakota, 151
get_parameter_sets
  Dakota::NonDSampling, 639, 640
get_pathext
  Dakota, 156
get_responses
  Dakota, 151
get_responses_from_dakota
  Dakota::Optimizer, 665
get_shared_data
  Dakota::SharedApproxData, 803
get_variable_bounds
  Dakota, 149
get_variable_bounds_from_dakota
  Dakota::Optimizer, 665
get_variables
  Dakota, 151
get_vbd_parameter_sets
  Dakota::Analyzer, 185
GetBestMOSolutions
  Dakota::JEGAOptimizer, 419
GetBestSOSolutions
  Dakota::JEGAOptimizer, 419
GetBestSolutions
Dakota::JEGAOptimizer, 419
GetDescription
  Dakota::JEGAOptimizer::Evaluator, 348
GetLongOpt, 372
  Dakota::GetLongOpt, 374
GetName
  Dakota::JEGAOptimizer::Evaluator, 348
GetNumberLinearConstraints
  Dakota::JEGAOptimizer::Evaluator, 347
GetNumberNonLinearConstraints
  Dakota::JEGAOptimizer::Evaluator, 347
getRmax
  Dakota, 153
getdist
  Dakota, 152
gnewton_set_recast
  Dakota::Iterator, 407
Graphics, 375
GridApplicInterface, 377
half_log_cov_det_gradient
  Dakota::ExperimentData, 357
half_log_cov_det_hessian
  Dakota::ExperimentData, 357
half_log_cov_determinant
  Dakota::ExperimentData, 357
hard_convergence_check
  Dakota::SurrBasedLocalMinimizer, 845
herbie
  Dakota::TestDriverInterface, 870
herbie1D
  Dakota::TestDriverInterface, 869
hessian
  Dakota::SurfpackApproximation, 837
HierarchSurrBasedLocalTraits, 379
HierarchSurrModel, 380
hom_constraint_eval
  Dakota::SurrBasedLocalMinimizer, 846
hom_objective_eval
  Dakota::SurrBasedLocalMinimizer, 846
id_vars_exact_compare
  Dakota, 154
import_points
  Dakota::DataFitSurrModel, 263
inactive_variables
  Dakota::Model, 475
inactiveVarsCompsTotals
  Dakota::SharedVariablesDataRep, 822
increment_grid_from_order
  Dakota::NonDPolynomialChaos, 616
increment_grid_preference
  Dakota::NonDCubature, 539
increment_order_and_grid
  Dakota::NonDPolynomialChaos, 616
increment_order_from_grid
  Dakota::NonDPolynomialChaos, 616
increment_parallel_configuration
  Dakota::ParallelLibrary, 686
increment_reference
  Dakota::NonDCubature, 540
init_communicators
  Dakota::Model, 473
  Dakota::ParallelLibrary, 686
init_communicators_checks
  Dakota::ApplicationInterface, 196
  Dakota::DirectApplicInterface, 325
  Dakota::ProcessHandleApplicInterface, 728
  Dakota::SysCallApplicInterface, 858
init_continuous_vars
  Dakota::DataTransformModel, 301
init_default_asv
  Dakota::ApplicationInterface, 197
init_iterator
  Dakota::IteratorScheduler, 412, 413
init_maps
  Dakota::RecastModel, 757
init_model
  Dakota::DataFitSurrModel, 264
init_mpi_comm
  Dakota::ParallelLibrary, 686
init_parameter_domain
  Dakota::NonDQUESOBayesCalibration, 626
initPreferred_env_path
  Dakota::WorkdirHelper, 899
init_serial
  Dakota::ApplicationInterface, 194
  Dakota::Model, 473
init_startup_path
  Dakota::WorkdirHelper, 899
initial_points
  Dakota::JEGAOptimizer, 420, 421
initial_taylor_series
  Dakota::NonDLLocalReliability, 589
initialize
  Dakota::NonDAdaptImpSampling, 521
  Dakota::WorkdirHelper, 896
initialize_class_data
  Dakota::NonDLLocalReliability, 590
Dakota::ExperimentData, 358
LoadAlgorithmConfig
   Dakota::JEGAOptimizer, 418
LoadDakotaResponses
   Dakota::JEGAOptimizer, 418
LoadProblemConfig
   Dakota::JEGAOptimizer, 418
LoadTheConstraints
   Dakota::JEGAOptimizer, 419
LoadTheDesignVariables
   Dakota::JEGAOptimizer, 418
LoadTheObjectiveFunctions
   Dakota::JEGAOptimizer, 419
LoadTheParameterDatabase
   Dakota::JEGAOptimizer, 418
local_eval_concurrency
   Dakota::Model, 471
   Dakota::NestedModel, 492
local_eval_synchronization
   Dakota::Model, 471
   Dakota::NestedModel, 492
local_recast_retrieve
   Dakota::Minimizer, 439
log_likelihood
   Dakota::NonDBayesCalibration, 535
logitTransform
   Dakota::NonDQUESOBayesCalibration, 627
lookup_by_val
   Dakota, 154
MPIManager, 478
MPIPackBuffer, 479
MPIUnpackBuffer, 481
main
   library_mode.cpp, 910
   main.cpp, 912
   restart_util.cpp, 913
main.cpp, 912
   fpinit_ASL, 912
   main, 912
make_variable_defaults
   Dakota::NIDRProblemDescDB, 500
manage_asv
   Dakota::Model, 477
manage_data_recastings
   Dakota::Model, 474
manage_linear_constraints
   Dakota::Constraints, 247
MandatoryValue
   Dakota::GetLongOpt, 374
map
   Dakota::ApplicationInterface, 194
map_domain
   Dakota::COLINApplication, 228
mappingInitialized
   Dakota::RandomFieldModel, 748
master_dynamic_schedule_analyses
   Dakota::ApplicationInterface, 196
master_dynamic_schedule_evaluations
   Dakota::ApplicationInterface, 197
master_dynamic_schedule_evaluations_nowait
   Dakota::ApplicationInterface, 199
master_dynamic_schedule_iterators
   Dakota::IteratorScheduler, 415
MatchesWC, 429
matlab_engine_run
   Dakota::MatlabInterface, 430
MatlabInterface, 429
maxEvalConcurrency
   Dakota::Iterator, 409
mc_api_run
   Dakota::TestDriverInterface, 870
MetaIterator, 431
mi_parallel_level
   Dakota::ParallelConfiguration, 673
mi_parallel_level_iterator
   Dakota::ParallelConfiguration, 673
mindist
   Dakota, 152
mindistindx
   Dakota, 153
Minimizer, 432
   Dakota::Minimizer, 437
MixedVarConstraints, 440
   Dakota::MixedVarConstraints, 441
MixedVariables, 441
   Dakota::MixedVariables, 442
mode_counts
   Dakota::NonDSampling, 642
Model, 443
   Dakota::Model, 468
modify_n2s
   Dakota::ScalingModel, 787
modify_s2n
   Dakota::ScalingModel, 787
mpi_debug_hold
   Dakota, 147
multilevel_control_variate_mc_Qcorr
   Dakota::NonDMultilevelSampling, 605
INDEX

multilevel_control_variate_mc_Ycorr
    Dakota::NonDMultilevelSampling, 605
multilevel_mc_Qsum
    Dakota::NonDMultilevelSampling, 604
multilevel_mc_Ysum
    Dakota::NonDMultilevelSampling, 604

NCSUTraits, 484
NIDRProblemDescDB, 495
NL2Res, 500
NL2SOLLeastSqTraits, 501
NLPSQLPTraits, 502
NLSSOLLeastSqTraits, 503
NOWPACBlackBoxEvaluator, 655
NOWPACOptimizer, 656
NOWPACTraits, 657
NPSOLTraits, 658
NUMBER_OF_FIELDS
    Dakota, 157
Name
    Dakota::JEGAOptimizer::Evaluator, 346
need_res_trans_byvars
    Dakota::ScalingModel, 787
neg_log_post_resp_mapping
    Dakota::NonDBayesCalibration, 535
NestedModel, 484
new_dataset
    Dakota::Graphics, 377
NoDBaseConstructor, 504
NomadOptimizer::Evaluator, 342
NomadTraits, 504
NonD, 506
NonDAdaptImpSampling, 517
    Dakota::NonDAdaptImpSampling, 520
NonDAdaptiveSampling, 521
    Dakota::NonDAdaptiveSampling, 526
NonDBayesCalibration, 527
    Dakota::NonDBayesCalibration, 533
NonDCalibration, 536
    Dakota::NonDCalibration, 537
NonDCubature, 537
    Dakota::NonDCubature, 539
NonDDREAMBayesCalibration, 540
    Dakota::NonDDREAMBayesCalibration, 542
NonDExpansion, 544
NonDGPImpSampling, 559
    Dakota::NonDGPImpSampling, 561
NonDGPSMABayesCalibration, 562
    Dakota::NonDGPSMABayesCalibration, 565
NonDGlobalEvidence, 550
    NonDGlobalInterval, 552
    NonDGlobalReliability, 555
    NonDGlobalSingleInterval, 558
    NonDIntegration, 566
        Dakota::NonDIntegration, 568
    NonInterval, 569
    NonDLHSEvidence, 572
    NonDLHSSingleInterval, 578
    NonDLocalEvidence, 579
    NonDLocalInterval, 580
    NonDLocalReliability, 583
    NonDLocalSingleInterval, 593
    NonDMultilevelPolynomialChaos, 594
        Dakota::NonDMultilevelPolynomialChaos, 596
    NonDMultilevelSampling, 597
        Dakota::NonDMultilevelSampling, 603
    NonDMultilevelStochCollocation, 606
        Dakota::NonDMultilevelStochCollocation, 607, 608
    NonDPOFDarts, 608
    NonDPolynomialChaos, 611
        Dakota::NonDPolynomialChaos, 615, 616
    NonDQUESOBayesCalibration, 621
        Dakota::NonDQUESOBayesCalibration, 625
    NonDQuadrature, 617
        Dakota::NonDQuadrature, 619, 620
    NonDRKDDarts, 629
    NonDReliability, 627
    NonDSampling, 632
        Dakota::NonDSampling, 637, 638
    NonDSParseGrid, 642
        Dakota::NonDSParseGrid, 644
    NonDStochCollocation, 645
        Dakota::NonDStochCollocation, 647
    NonDWASABIBayesCalibration, 648
        Dakota::NonDWASABIBayesCalibration, 651
NonlinearCGOptimizer, 651
NonlinearCGTraits, 654
num_samples
    Dakota::Analyzer, 182
    Dakota::DADACEDesignCompExp, 320
    Dakota::FSUDesignCompExp, 366
    Dakota::NonDCubature, 540
    Dakota::NonDQuadrature, 620
    Dakota::NonDSampling, 639
    Dakota::NonDSParseGrid, 645
    Dakota::PSUADEDesignCompExp, 740
objective
  Dakota::Minimizer, 438
objective_gradient
  Dakota::Minimizer, 439
objective_hessian
  Dakota::Minimizer, 439
objective_reduction
  Dakota::Optimizer, 666
operator<<
  Dakota::CommandShell, 233
operator()
  Dakota::BootstrapSampler, 221
  Dakota::BootstrapSampler< Teuchos::SerialDenseMatrix<OrdinalType, ScalarType>>, 222
  Dakota::LabelsWriter, 422
  Dakota::TabularWriter, 860
  Dakota::TabularWriter, 861
operator=
  Dakota::Approximation, 207
  Dakota::Constraints, 247
  Dakota::Environment, 341
  Dakota::Interface, 393
  Dakota::Iterator, 404
  Dakota::Model, 469
  Dakota::ProblemDescDB, 720
  Dakota::SharedApproxData, 803
operator==
  Dakota, 149, 153
  OptDartsOptimizer, 659
  OptDartsTraits, 662
  OptType
    Dakota::GetLongOpt, 374
  Optimizer, 662
  OptionalValue
    Dakota::GetLongOpt, 374
  output_helper
    Dakota::CommandLineHandler, 232
  OutputManager, 667
    Dakota::OutputManager, 670
  OutputWriter, 670
PMA2_constraint_eval
  Dakota::NonDLocalReliability, 589
PMA_constraint_eval
  Dakota::NonDLocalReliability, 589
PMA_objective_eval
  Dakota::NonDLocalReliability, 588
PRPMultiIndexCache
  Dakota, 146
PRPMultiIndexQueue
  Dakota, 146
PSUADEDesignCompExp
  Dakota::PSUADEDesignCompExp, 737
PStudyDACE, 735
parallel_input
  library_mode.cpp, 911
parallel_interface_plugin
  library_mode.cpp, 910
ParallelConfiguration, 671
ParallelDirectApplicInterface, 673
ParallelLevel, 675
ParallelLibrary, 677
  Dakota::ParallelLibrary, 685
ParamResponsePair, 687
  Dakota::ParamResponsePair, 689
ParamStudy, 690
parse
  Dakota::Environment, 342
  Dakota::GetLongOpt, 374
parse_inputs
  Dakota::ProblemDescDB, 720
parse_sigma_types
  Dakota::ExperimentData, 358
partial_corr
  Dakota::SensAnalysisGlobal, 793
  partial_prp_equality, 696
  partial_prp_hash, 696
partition
  Dakota::IteratorScheduler, 414
  PebblBranchSub, 697
  PebblBranching, 696
  PebblTraits, 699
PecosApproximation, 700
peer_dynamic_schedule_evaluations
  Dakota::ApplicationInterface, 198
  peer_dynamic_schedule_evaluations_nowait
    Dakota::ApplicationInterface, 200
  peer_static_schedule_evaluations
    Dakota::ApplicationInterface, 198
  peer_static_schedule_evaluations_nowait
    Dakota::ApplicationInterface, 199
penaltyMerit
  Dakota::SurrBasedMinimizer, 851
perform_analysis
  Dakota, 153
perform_evaluation_impl
  Dakota::COLINAApplication, 227
PerformIterations
  Dakota::JEGAOptimizer::Driver, 332
po_which
  Dakota::WorkdirHelper, 898
poly_prod
  Dakota::TestDriverInterface, 868
pop
  Dakota::Approximation, 208
  Dakota::PecosApproximation, 704
pop_approximation
  Dakota::ApproximationInterface, 214
pop_output_tag
  Dakota::OutputManager, 670
post_process
  Dakota::ProblemDescDB, 721
post_run
  Dakota::Analyzer, 183
  Dakota::DDACEDesignCompExp, 320
  Dakota::FSUDesignCompExp, 365
  Dakota::Iterator, 405
  Dakota::LeastSq, 425
  Dakota::MetaIterator, 432
  Dakota::Minimizer, 437
  Dakota::NonDMultilevelSampling, 603
  Dakota::NonDRKDarts, 631
  Dakota::Optimizer, 665
  Dakota::ParamStudy, 695
  Dakota::PSUADEDesignCompExp, 739
  Dakota::SurrBasedLocalMinimizer, 844
pre_output
  Dakota::Analyzer, 184
pre_run
  Dakota::Analyzer, 183
  Dakota::ConcurrentMetaIterator, 236
  Dakota::DDACEDesignCompExp, 320
  Dakota::FSUDesignCompExp, 365
  Dakota::Iterator, 404
  Dakota::NonD, 515
  Dakota::NonDMultilevelSampling, 603
  Dakota::NonDRKDarts, 631
  Dakota::NonDSampling, 639
  Dakota::PSUADEDesignCompExp, 739
  Dakota::SurrBasedLocalMinimizer, 844
prepare_process_environment
  Dakota::ProcessApplicInterface, 725
prepend_path_item
  Dakota::WorkdirHelper, 898
prepend_preferred_env_path
  Dakota::WorkdirHelper, 896
primaryRespReducer
  Dakota::Optimizer, 666
  primaryResp_weighter
  Dakota::LeastSq, 426
print_level_map
  Dakota::NonD, 517
print_level_mappings
  Dakota::NonD, 515
print_restart
  Dakota, 155
print_restart_pdb
  Dakota, 155
print_restart_tabular
  Dakota, 155
print_results
  Dakota::Analyzer, 184
  Dakota::ConcurrentMetaIterator, 236
  Dakota::Iterator, 406
  Dakota::LeastSq, 425
  Dakota::NonDAdaptiveSampling, 526
  Dakota::NonDBayesCalibration, 534
  Dakota::NonDG2010, 557
  Dakota::NonDG2010BayesCalibration, 566
  Dakota::NonDMultilevelSampling, 604
  Dakota::NonDQUESOBayesCalibration, 625
  Dakota::NonDQUESOBayesCalibration, 651
  Dakota::Optimizer, 665
  Dakota::PStudyDACE, 736
  Dakota::RichExtrapVerification, 781
  Dakota::SeqHybridMetaIterator, 796
  Dakota::SurrBasedLocalMinimizer, 849
  Dakota::Verification, 888
print_sobol_indices
  Dakota::Analyzer, 186
prior_density
  Dakota::NonDDREAMBayesCalibration, 542
prior_mean
  Dakota::NonDBayesCalibration, 534
prior_proposal_covariance
  Dakota::NonDQUESOBayesCalibration, 626
prior_sample
  Dakota::NonDDREAMBayesCalibration, 543
prior_variance
  Dakota::NonDBayesCalibration, 534
probDescDB
  Dakota::Iterator, 408
  Dakota::Model, 478
probability
  Dakota::NonDLocalReliability, 592
ProbabilityTransformModel, 704
problem_size
Dakota::NonDDREAMBayesCalibration, 542
problem_value
Dakota::NonDDREAMBayesCalibration, 542
ProblemDescDB, 711
Dakota::ProblemDescDB, 720
ProcessApplicInterface, 722
ProcessHandleApplicInterface, 726
Dakota::ProcessHandleApplicInterface, 728
ProgramOptions, 730
ptmInstance
Dakota::ProbabilityTransformModel, 711
push
Dakota::Approximation, 208
Dakota::PecosApproximation, 704
push_approximation
Dakota::ApproximationInterface, 215
push_output_tag
Dakota::ParallelLibrary, 685
python_convert_int
Dakota::PythonInterface, 742
PythonInterface, 740
qr
Dakota, 148
qr_rsolve
Dakota, 148
QuesoJointPdf
Dakota::QuesoJointPdf, 743
QuesoJointPdf < V, M >, 742
QuesoVectorRV
Dakota::QuesoVectorRV, 743
QuesoVectorRV < V, M >, 743
RIA_constraint_eval
Dakota::NonDLocalReliability, 588
RIA_objective_eval
Dakota::NonDLocalReliability, 588
RandomFieldModel, 744
rawResponseMap
Dakota::Interface, 395
read
Dakota::ParamResponsePair, 689
read_challenge_points
Dakota::ApproximationInterface, 215
read_core
Dakota::MixedVariables, 442
Dakota::RelaxedVariables, 762
read_neutral
Dakota, 155
read_results_file
Dakota::ProcessApplicInterface, 726
read_tabular
Dakota::MixedVariables, 442
Dakota::RelaxedVariables, 762
read_variables_responses
Dakota::Analyzer, 185
rebuild
Dakota::Approximation, 208
Dakota::PecosApproximation, 703
rebuild_approximation
Dakota::ApproximationInterface, 214
RecastModel, 749
Dakota::RecastModel, 756
RecordResponses
Dakota::JEGAOptimizer::Evaluator, 347
recover_model
Dakota::ExperimentData, 356
recursive_copy
Dakota::WorkdirHelper, 898
recv
Dakota::APPSEvalMgr, 217
reduce_model
Dakota::Optimizer, 666
ReducedBasis, 759
register_signal_handlers
Dakota, 147
RelaxedVarConstraints, 760
Dakota::RelaxedVarConstraints, 760
RelaxedVariables, 761
Dakota::RelaxedVariables, 762
repair_restart
Dakota, 155
requestVector
Dakota::ActiveSet, 167
reset_process_environment
Dakota::ProcessApplicInterface, 726
reshape
Dakota::Constraints, 247
residuals_per_multiplier
Dakota::ExperimentData, 358
resize_best_resp_array
Dakota::Minimizer, 439
resize_best_vars_array
Dakota::Minimizer, 439
resolve_inputs
Dakota::ParallelLibrary, 686
resolve_samples_symbols
Dakota::DDACEDesignCompExp, 321
INDEX

serial_input
  library_mode.cpp, 910
serial_interface_plugin
  library_mode.cpp, 909
SerialDirectApplicInterface, 797
serve_analyses_asynch
  Dakota::ProcessHandleApplicInterface, 729
serve_analyses_synch
  Dakota::ApplicationInterface, 196
serve_evaluations
  Dakota::ApplicationInterface, 195
serve_evaluations_asynch
  Dakota::ApplicationInterface, 201
serve_evaluations_asynch_peer
  Dakota::ApplicationInterface, 201
serve_evaluations_synch
  Dakota::ApplicationInterface, 201
serve_evaluations_synch_peer
  Dakota::ApplicationInterface, 201
serve_iterators
  Dakota::IteratorScheduler, 415
set_best_responses
  Dakota, 150
set_communicators_checks
  Dakota::ApplicationInterface, 196
  Dakota::DirectApplicInterface, 325
  Dakota::ProcessHandleApplicInterface, 728
  Dakota::SysCallApplicInterface, 858
set_compare
  Dakota, 154
set_ip_options
  Dakota::NonDQUESOBayesCalibration, 626
set_iterator
  Dakota::IteratorScheduler, 413
set_mapping
  Dakota::ActiveSubspaceModel, 174
  Dakota::AdaptedBasisModel, 178
  Dakota::DataTransformModel, 301
set_preferred_path
  Dakota::WorkdirHelper, 898
set_problem
  Dakota::COLINApplication, 226
set_u_to_x_mapping
  Dakota::NonD, 517
  Dakota::ProbabilityTransformModel, 711
set_variables
  Dakota, 150
shape
  Dakota::Constraints, 247

SharedApproxData, 798
  Dakota::SharedApproxData, 802
  SharedPecosApproxData, 804
  SharedResponseData, 806
  SharedResponseDataRep, 809
  SharedSurfpackApproxData, 811
  Dakota::SharedSurfpackApproxData, 812
  SharedVariablesData, 812
  SharedVariablesDataRep, 818
shubert1D
  Dakota::TestDriverInterface, 870
simple_corr
  Dakota::SensAnalysisInterface, 792
SimulationModel, 823
SimulationResponse, 826
smooth_herbie
  Dakota::TestDriverInterface, 870
  smooth_herbie1D
  Dakota::TestDriverInterface, 869
solution_level_index
  Dakota::Model, 471
solution_levels
  Dakota::Model, 471
spawn_analysis_to_shell
  Dakota::SysCallApplicInterface, 859
spawn_evaluation_impl
  Dakota::COLINApplication, 227
spawn_evaluation_to_shell
  Dakota::SysCallApplicInterface, 858
spawn_input_filter_to_shell
  Dakota::SysCallApplicInterface, 858
spawn_output_filter_to_shell
  Dakota::SysCallApplicInterface, 859
SpawnApplicInterface, 833
split_filenames
  Dakota::ProgramOptions, 735
splitWildcard
  Dakota::WorkdirHelper, 897
start_dakota_heartbeat
  Dakota, 153
start_grid_computing
  Dakota, 153
steady_state_diffusion_1d
  Dakota::TestDriverInterface, 868
steel_column_cost
  Dakota::TestDriverInterface, 869
stop_evaluation_servers
  Dakota::ApplicationInterface, 195
stop_grid_computing
INDEX

Dakota, 153
  subModel
    Dakota::NestedModel, 495
submit
  Dakota::APPSEvalMgr, 217
subordinate_iterator
  Dakota::Model, 469
subordinate_model
  Dakota::Model, 469
subordinate_models
  Dakota::Model, 472
SurfpackApproximation, 834
  Dakota::SurfpackApproximation, 836
SurrBasedGlobalMinimizer, 838
SurrBasedGlobalTraits, 839
SurrBasedLocalMinimizer, 840
SurrBasedMinimizer, 847
surrogate_model
  Dakota::Model, 469
SurrogateModel, 851
surrogates_to_surf_data
  Dakota::SurfpackApproximation, 837
tsvd
  Dakota, 148
symmetric_eigenvalue_decomposition
  Dakota, 152
synchronize
  Dakota::ApplicationInterface, 195
  synchronize_derivatives
  Dakota::Model, 476
  synchronize_nowait
  Dakota::ApplicationInterface, 195
synchronous_local_analyses
  Dakota::ProcessApplicInterface, 725
synchronous_local_analysis
  Dakota::DirectApplicInterface, 325
  Dakota::GridApplicInterface, 379
  Dakota::ProcessHandleApplicInterface, 728
  Dakota::SysCallApplicInterface, 858
synchronous_local_evaluations
  Dakota::ApplicationInterface, 199
SysCallApplicInterface, 856
test_local_evaluations
  Dakota::SysCallApplicInterface, 857
  SIM::ParallelDirectApplicInterface, 674
  SIM::SerialDirectApplicInterface, 798
TestDriverInterface, 864
ToDoubleMatrix
  Dakota::JEGAOptimizer, 420
tokenize_env_path
  Dakota::WorkdirHelper, 899
TrackerHTTP, 871
TrackerHTTP::Server, 798
TraitsBase, 873
transform_response_map
  Dakota::DataTransformModel, 301
transform_samples
  Dakota::NonDSampling, 639
trendOrder
  Dakota::GaussProcApproximation, 371
truth_model
  Dakota::Model, 470
uncentered_to_centered
  Dakota::NonDMultilevelSampling, 606
uncertain_vars_to_subspace
  Dakota::ActiveSubspaceModel, 173
  Dakota::AdaptedBasisModel, 177
update_approximation
  Dakota::ApproximationInterface, 213
  Dakota::DataFitSurrModel, 260, 261
update_augmented_lagrange_multipliers
  Dakota::SurrBasedMinimizer, 850
update_filter
  Dakota::SurrBasedMinimizer, 850
update_from_model
  Dakota::DataFitSurrModel, 265
  Dakota::RecastModel, 757
update_from_subordinate_model
  Dakota::Model, 470
update_lagrange_multipliers
  Dakota::SurrBasedMinimizer, 849
update_level_data
  Dakota::NonDLocalReliability, 591
update_model
  Dakota::DataFitSurrModel, 265
update_mpp_search_data
  Dakota::NonDLocalReliability, 591
update_penalty
  Dakota::SurrBasedLocalMinimizer, 845
update_quasi_hessians
  Dakota::Model, 476
update_response
  Dakota::Model, 476
usage
  Dakota::GetLongOpt, 375
UsageTracker, 875
  Dakota::UsageTracker, 876
useDerivs
  Dakota::NonDExpansion, 550
user_proposal_covariance
  Dakota::NonDQUESOBayesCalibration, 626
uses_method
  Dakota::Iterator, 406
VLUncertainInt
  Dakota, 159
VLUncertainReal
  Dakota, 159
VLUncertainStr
  Dakota, 160
VLint, 888
VLreal, 889
VLstr, 889
VPSApproximation, 890
VPSinstance
  Dakota::VPSApproximation, 893
VPSmodel_apply
  Dakota::VPSApproximation, 893
Valueless
  Dakota::GetLongOpt, 374
values_to_ranks
  Dakota::SensAnalysisGlobal, 792
Var_check, 876
var_mp_check_bound
  Dakota, 162
var_mp_check_cau
  Dakota, 160
var_mp_check_ceu
  Dakota, 161
var_mp_check_cv
  Dakota, 160
var_mp_check_daui
  Dakota, 161
var_mp_check_daur
  Dakota, 161
var_mp_check_daus
  Dakota, 161
var_mp_check_deui
  Dakota, 161
var_mp_check_deur
  Dakota, 162
var_mp_check_deus
  Dakota, 161
var_mp_check_dset
  Dakota, 160
var_mp_drange
  Dakota, 162
Var_rcheck, 877
Variables, 877
variables_expand
  Dakota::DataTransformModel, 300
variables_resize
  Dakota::RandomFieldModel, 747
variables_scaler
  Dakota::ScalingModel, 787
variables_to_sample
  Dakota::Analyzer, 186
  Dakota::NonDSampling, 640
variablesCompsTotals
  Dakota::SharedVariablesDataRep, 822
vars_mapping
  Dakota::ActiveSubspaceModel, 173
  Dakota::AdaptedBasisModel, 178
  Dakota::RandomFieldModel, 747
vars_u_to_x_mapping
  Dakota::NonD, 516
  Dakota::ProbabilityTransformModel, 710
vars_x_to_u_mapping
  Dakota::NonD, 516
  Dakota::ProbabilityTransformModel, 710
Verification, 887
view_aleatory_uncertain_counts
  Dakota::NonDSampling, 641
view_design_counts
  Dakota::NonDSampling, 641
view_epistemic_uncertain_counts
  Dakota::NonDSampling, 641
view_uncertain_counts
  Dakota::NonDSampling, 641
volumetric_quality
  Dakota::PStudyDACE, 736
wait_local_evaluations
  Dakota::SysCallApplicInterface, 857
weight_model
  Dakota::LeastSq, 426
which
  Dakota::WorkdirHelper, 896
WorkdirHelper, 894
write
  Dakota::ParamResponsePair, 689
INDEX

write_ordered
  Dakota, 151, 152