Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis

Version 5.4 Developers Manual

Brian M. Adams, Mohamed S. Ebeida, Michael S. Eldred, John D. Jakeman, Laura P. Swiler
Optimization and Uncertainty Quantification Department

William J. Bohnhoff
Radiation Transport Department

Keith R. Dalbey
Mission Analysis and Simulation Department

John P. Eddy
System Readiness and Sustainment Technologies Department

Kenneth T. Hu
Validation and Uncertainty Quantification Department

Dena M. Vigil
Multiphysics Simulation Technologies Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185

Lara E. Bauman, Patricia D. Hough
Quantitative Modeling and Analysis Department

Sandia National Laboratories
P.O. Box 969
Livermore, CA 94551
Abstract

The Dakota (Design Analysis Kit for Optimization and Terascale Applications) 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 serves as a developers manual for the Dakota software and describes the Dakota class hierarchies and their interrelationships. It derives directly from annotation of the source code and provides detailed class documentation, including all member functions and attributes.
# Contents

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

2 Coding Style Guidelines and Conventions  
   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  
   3.1 Modify dakota.input.nspec ............................................... 27  
   3.2 Rebuild generated files ................................................... 28  
   3.3 Update NIDRProblemDescDB.cpp in Dakota/src .................. 28  
   3.4 Update ProblemDescDB.cpp in Dakota/src .......................... 30  
   3.5 Update Corresponding Data Classes .................................. 31  
   3.6 Use get_<data_type>() Functions .................................... 31  
   3.7 Update the Documentation .............................................. 31  

4 Understanding Iterator Flow .................................................. 33  

5 Interfacing with Dakota as a Library ...................................... 35  
   5.1 Introduction ..................................................................... 35
CONTENTS

5.2 Quick start: examples and test code ........................................ 35
5.3 Comparison to main.cpp ..................................................... 36
5.4 Problem database population ............................................... 37
5.5 Instantiating the strategy .................................................... 39
5.6 Defining the direct application interface ............................... 40
5.7 Additional updates ............................................................ 42
5.8 Executing the strategy ....................................................... 42
5.9 Retrieving data after a run .................................................. 42
5.10 Linking against the Dakota library ...................................... 43
5.11 Summary ............................................................................ 44

6 Performing Function Evaluations ............................................ 45
6.1 Synchronous function evaluations ........................................ 45
6.2 Asynchronous function evaluations ...................................... 45
6.3 Analyses within each function evaluation .............................. 46

7 Working with Variable Containers and Views .......................... 47
7.1 Storage in Variables .......................................................... 47
7.2 Storage in SharedVariablesData .......................................... 48
7.3 Active and inactive views .................................................... 49

8 Namespace Index ..................................................................... 51
8.1 Namespace List .................................................................... 51

9 Class Index ........................................................................... 53
9.1 Class Hierarchy .................................................................. 53

10 Class Index ......................................................................... 57
10.1 Class List ......................................................................... 57

11 File Index ............................................................................ 63
11.1 File List ........................................................................... 63

12 Namespace Documentation .................................................. 65
12.1 Dakota Namespace Reference ......................................... 65
12.2 SIM Namespace Reference .............................................. 240
# 13 Class Documentation

<table>
<thead>
<tr>
<th>Section</th>
<th>Class Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1</td>
<td>ActiveSet</td>
<td>241</td>
</tr>
<tr>
<td>13.2</td>
<td>Analyzer</td>
<td>245</td>
</tr>
<tr>
<td>13.3</td>
<td>ApplicationInterface</td>
<td>251</td>
</tr>
<tr>
<td>13.4</td>
<td>Approximation</td>
<td>269</td>
</tr>
<tr>
<td>13.5</td>
<td>ApproximationInterface</td>
<td>279</td>
</tr>
<tr>
<td>13.6</td>
<td>APPSEvalMgr</td>
<td>286</td>
</tr>
<tr>
<td>13.7</td>
<td>APPSOptimizer</td>
<td>289</td>
</tr>
<tr>
<td>13.8</td>
<td>BaseConstructor</td>
<td>293</td>
</tr>
<tr>
<td>13.9</td>
<td>BiStream</td>
<td>294</td>
</tr>
<tr>
<td>13.10</td>
<td>BoStream</td>
<td>297</td>
</tr>
<tr>
<td>13.11</td>
<td>COLINApplication</td>
<td>300</td>
</tr>
<tr>
<td>13.12</td>
<td>COLINOptimizer</td>
<td>304</td>
</tr>
<tr>
<td>13.13</td>
<td>CollaborativeHybridStrategy</td>
<td>309</td>
</tr>
<tr>
<td>13.14</td>
<td>CommandLineHandler</td>
<td>311</td>
</tr>
<tr>
<td>13.15</td>
<td>CommandShell</td>
<td>314</td>
</tr>
<tr>
<td>13.16</td>
<td>ConcurrentStrategy</td>
<td>316</td>
</tr>
<tr>
<td>13.17</td>
<td>CONMINOptimizer</td>
<td>319</td>
</tr>
<tr>
<td>13.18</td>
<td>Constraints</td>
<td>327</td>
</tr>
<tr>
<td>13.19</td>
<td>DataFitSurrModel</td>
<td>340</td>
</tr>
<tr>
<td>13.20</td>
<td>DataInterface</td>
<td>354</td>
</tr>
<tr>
<td>13.21</td>
<td>DataMethod</td>
<td>356</td>
</tr>
<tr>
<td>13.22</td>
<td>DataMethodRep</td>
<td>358</td>
</tr>
<tr>
<td>13.23</td>
<td>DataModel</td>
<td>374</td>
</tr>
<tr>
<td>13.24</td>
<td>DataModelRep</td>
<td>376</td>
</tr>
<tr>
<td>13.25</td>
<td>DataResponses</td>
<td>382</td>
</tr>
<tr>
<td>13.26</td>
<td>DataResponsesRep</td>
<td>384</td>
</tr>
<tr>
<td>13.27</td>
<td>DataStrategy</td>
<td>389</td>
</tr>
<tr>
<td>13.28</td>
<td>DataStrategyRep</td>
<td>391</td>
</tr>
<tr>
<td>13.29</td>
<td>DataVariables</td>
<td>394</td>
</tr>
<tr>
<td>13.30</td>
<td>DataVariablesRep</td>
<td>396</td>
</tr>
<tr>
<td>13.31</td>
<td>DDACEDesignCompExp</td>
<td>409</td>
</tr>
<tr>
<td>13.32</td>
<td>DirectApplicInterface</td>
<td>414</td>
</tr>
<tr>
<td>13.33</td>
<td>DiscrepancyCorrection</td>
<td>420</td>
</tr>
<tr>
<td>Class Reference</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>13.34 DOTOptimizer Class Reference</td>
<td>425</td>
<td></td>
</tr>
<tr>
<td>13.35 Driver Class Reference</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>13.36 EffGlobalMinimizer Class Reference</td>
<td>432</td>
<td></td>
</tr>
<tr>
<td>13.37 EfficientSubspaceMethod Class Reference</td>
<td>435</td>
<td></td>
</tr>
<tr>
<td>13.38 EmbeddedHybridStrategy Class Reference</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>13.39 Evaluator Class Reference</td>
<td>442</td>
<td></td>
</tr>
<tr>
<td>13.40 Evaluator Class Reference</td>
<td>444</td>
<td></td>
</tr>
<tr>
<td>13.41 EvaluatorCreator Class Reference</td>
<td>451</td>
<td></td>
</tr>
<tr>
<td>13.42 ExperimentData Class Reference</td>
<td>453</td>
<td></td>
</tr>
<tr>
<td>13.43 ForkApplicInterface Class Reference</td>
<td>454</td>
<td></td>
</tr>
<tr>
<td>13.44 FSUDesignCompExp Class Reference</td>
<td>457</td>
<td></td>
</tr>
<tr>
<td>13.45 GaussProcApproximation Class Reference</td>
<td>462</td>
<td></td>
</tr>
<tr>
<td>13.46 GetLongOpt Class Reference</td>
<td>469</td>
<td></td>
</tr>
<tr>
<td>13.47 Graphics Class Reference</td>
<td>473</td>
<td></td>
</tr>
<tr>
<td>13.48 GridApplicInterface Class Reference</td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>13.49 HierarchSurrModel Class Reference</td>
<td>480</td>
<td></td>
</tr>
<tr>
<td>13.50 HybridStrategy Class Reference</td>
<td>485</td>
<td></td>
</tr>
<tr>
<td>13.51 Interface Class Reference</td>
<td>487</td>
<td></td>
</tr>
<tr>
<td>13.52 Iterator Class Reference</td>
<td>498</td>
<td></td>
</tr>
<tr>
<td>13.53 JEGAOptimizer Class Reference</td>
<td>513</td>
<td></td>
</tr>
<tr>
<td>13.54 LeastSq Class Reference</td>
<td>521</td>
<td></td>
</tr>
<tr>
<td>13.55 MatlabInterface Class Reference</td>
<td>526</td>
<td></td>
</tr>
<tr>
<td>13.56 Minimizer Class Reference</td>
<td>528</td>
<td></td>
</tr>
<tr>
<td>13.57 MixedVarConstraints Class Reference</td>
<td>542</td>
<td></td>
</tr>
<tr>
<td>13.58 MixedVariables Class Reference</td>
<td>544</td>
<td></td>
</tr>
<tr>
<td>13.59 Model Class Reference</td>
<td>546</td>
<td></td>
</tr>
<tr>
<td>13.60 MPIPackBuffer Class Reference</td>
<td>581</td>
<td></td>
</tr>
<tr>
<td>13.61 MPIUnpackBuffer Class Reference</td>
<td>584</td>
<td></td>
</tr>
<tr>
<td>13.62 NCSUOptimizer Class Reference</td>
<td>587</td>
<td></td>
</tr>
<tr>
<td>13.63 NestedModel Class Reference</td>
<td>591</td>
<td></td>
</tr>
<tr>
<td>13.64 NIDRProblemDescDB Class Reference</td>
<td>601</td>
<td></td>
</tr>
<tr>
<td>13.65 NL2Res Struct Reference</td>
<td>606</td>
<td></td>
</tr>
<tr>
<td>13.66 NL2SOLLeastSq Class Reference</td>
<td>607</td>
<td></td>
</tr>
<tr>
<td>13.67 NLPQLPOptimizer Class Reference</td>
<td>610</td>
<td></td>
</tr>
</tbody>
</table>
13.68NLSSOLLeastSq Class Reference .......................... 616
13.69NoDBaseConstructor Struct Reference .................. 618
13.70NomadOptimizer Class Reference ....................... 619
13.71NonD Class Reference .................................... 622
13.72NonDAadaptImpSampling Class Reference .............. 634
13.73NonDAadaptiveSampling Class Reference .............. 638
13.74NonDBayesCalibration Class Reference ................ 644
13.75NonDCalibration Class Reference ....................... 646
13.76NonDCubature Class Reference ......................... 649
13.77NonDDREAMBayesCalibration Class Reference ........ 652
13.78NonDExpansion Class Reference ....................... 657
13.79NonDGlobalEvidence Class Reference .................. 665
13.80NonDGlobalInterval Class Reference ................... 667
13.81NonDGlobalReliability Class Reference ............... 671
13.82NonDGlobalSingleInterval Class Reference ........... 674
13.83NonDGPImpSampling Class Reference .................. 676
13.84NonDGPMSABayesCalibration Class Reference .......... 680
13.85NonDIncremLHSSampling Class Reference .......... ... 684
13.86NonDIntegration Class Reference ....................... 687
13.87NonDInterval Class Reference ......................... 691
13.88NonDLHSEvidence Class Reference ..................... 694
13.89NonDLHSInterval Class Reference ..................... 696
13.90NonDLHSSampling Class Reference ...................... 698
13.91NonDLHSSingleInterval Class Reference .............. 701
13.92NonDLocalEvidence Class Reference ................... 703
13.93NonDLocalInterval Class Reference ................... 705
13.94NonDLocalReliability Class Reference ................. 708
13.95NonDLocalSingleInterval Class Reference .......... 720
13.96NonDPOFDarts Class Reference ......................... 722
13.97NonDPolynomialChaos Class Reference ................. 725
13.98NonDQuadrature Class Reference ...................... 730
13.99NonDQUESOBayesCalibration Class Reference ....... 735
13.10NonDReliability Class Reference ...................... 738
13.10NonDSampling Class Reference ......................... 740
| 13.10 | NonDSparseGrid Class Reference | 748 |
| 13.10 | NonDStochCollocation Class Reference | 752 |
| 13.10 | nonlinearCGOptimizer Class Reference | 755 |
| 13.10 | NonPSOLOptimizer Class Reference | 759 |
| 13.10 | Optimizer Class Reference | 763 |
| 13.10 | ParallelConfiguration Class Reference | 769 |
| 13.10 | ParallelDirectApplicInterface Class Reference | 771 |
| 13.10 | ParallelLevel Class Reference | 773 |
| 13.11 | ParallelLibrary Class Reference | 777 |
| 13.11 | ParamResponsePair Class Reference | 792 |
| 13.11 | ParamStudy Class Reference | 796 |
| 13.11 | partial_prp_equality Struct Reference | 802 |
| 13.11 | partial_prp_hash Struct Reference | 803 |
| 13.11 | PecosApproximation Class Reference | 804 |
| 13.11 | ProblemDescDB Class Reference | 812 |
| 13.11 | ProcessApplicInterface Class Reference | 824 |
| 13.11 | ProcessHandleApplicInterface Class Reference | 829 |
| 13.11 | StudyDACE Class Reference | 834 |
| 13.12 | PSUADesignCompExp Class Reference | 837 |
| 13.12 | PythonInterface Class Reference | 841 |
| 13.12 | RecastBaseConstructor Struct Reference | 844 |
| 13.12 | RecastModel Class Reference | 845 |
| 13.12 | RelaxedVarConstraints Class Reference | 856 |
| 13.12 | RelaxedVariables Class Reference | 858 |
| 13.12 | Response Class Reference | 860 |
| 13.12 | ResponseRep Class Reference | 866 |
| 13.12 | ResultsDBAny Class Reference | 874 |
| 13.12 | ResultsEntry<StoredType> Class Template Reference | 877 |
| 13.13 | ResultsID Class Reference | 879 |
| 13.13 | ResultsManager Class Reference | 881 |
| 13.13 | ResultsNames Class Reference | 884 |
| 13.13 | RichExtrapVerification Class Reference | 886 |
| 13.13 | ScilabInterface Class Reference | 889 |
| 13.13 | SensAnalysisGlobal Class Reference | 891 |
CONTENTS

13.136 SequentialHybridStrategy Class Reference ................................................. 893
13.137 SerialDirectApplicInterface Class Reference ............................................. 898
13.138 SharedVariablesData Class Reference ......................................................... 900
13.139 SharedVariablesDataRep Class Reference .................................................... 905
13.140 SingleMethodStrategy Class Reference ....................................................... 909
13.141 SingleModel Class Reference ...................................................................... 911
13.142 SNLLBase Class Reference ......................................................................... 914
13.143 SNLLLeastSq Class Reference .................................................................... 917
13.144 SNLLOptimizer Class Reference .................................................................. 922
13.145 SOLBase Class Reference .......................................................................... 930
13.146 SpawnApplicInterface Class Reference ....................................................... 933
13.147 Strategy Class Reference ......................................................................... 935
13.148 SurfpackApproximation Class Reference ...................................................... 944
13.149 SurrBasedGlobalMinimizer Class Reference ................................................ 949
13.150 SurrBasedLocalMinimizer Class Reference .................................................. 951
13.151 SurrBasedMinimizer Class Reference .......................................................... 960
13.152 SurrogateModel Class Reference ................................................................ 966
13.153 SysCallApplicInterface Class Reference ...................................................... 971
13.154 TANA3Approximation Class Reference ......................................................... 976
13.155 TaylorApproximation Class Reference ......................................................... 979
13.156 TestDriverInterface Class Reference ........................................................... 981
13.157 TrackerHTTP Class Reference ..................................................................... 987
13.158 Variables Class Reference .......................................................................... 990
13.159 Verification Class Reference ....................................................................... 1003

14 File Documentation ......................................................................................... 1005
14.1 dakota_dll_api.cpp File Reference ................................................................. 1005
14.2 dakota_dll_api.h File Reference ................................................................... 1007
14.3 dakota_tabular_io.hpp File Reference ............................................................. 1009
14.4 JEGAOptimizer.cpp File Reference ................................................................. 1011
14.5 JEGAOptimizer.hpp File Reference .................................................................. 1012
14.6 library_mode.cpp File Reference .................................................................. 1013
14.7 library_split.cpp File Reference ................................................................... 1015
14.8 main.cpp File Reference .............................................................................. 1016
Chapter 1

Dakota Developers Manual

Author:

1.1 Introduction

The Dakota (Design Analysis Kit for Optimization and Terascale Applications) toolkit provides a flexible, extensible interface between analysis codes and iteration methods. 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, and for more details on Dakota features and capabilities, refer to the Users Manual.

1.2 Overview of Dakota

In Dakota, the strategy creates and manages iterators and models. In the simplest case, the strategy creates a single iterator and a single model and executes the iterator on the model to perform a single study. In a more advanced case, a hybrid optimization strategy might manage a global optimizer operating on a low-fidelity model in coordination with a local optimizer operating on a high-fidelity model. And on the high end, a surrogate-based optimization under uncertainty strategy would employ an uncertainty quantification iterator nested within
an optimization iterator and would employ truth models layered 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 strategy, 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 Strategies

Class hierarchy: **Strategy**.

Strategies provide a control layer for creation and management of iterators and models. Specific strategies include:

- **SingleMethodStrategy**: the simplest strategy. A single iterator is run on a single model to perform a single study.

- **HybridStrategy**: hybrid minimization using a set of iterators employing a corresponding set of models of varying fidelity. Coordination approaches among the iterators include collaborative, embedded, and sequential approaches, as embodied in the CollaborativeHybridStrategy, EmbeddedHybridStrategy, and SequentialHybridStrategy derived classes.

- **ConcurrentStrategy**: two similar algorithms are available: (1) multi-start iteration from several different starting points, and (2) pareto set optimization for several different multiobjective weightings. Employs a single iterator with a single model, but runs multiple instances of the iterator concurrently for different settings within the model.

### 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, non-linear least squares, design of experiments, and parameter studies. The hierarchy is divided into Minimizer and Analyzer branches. The Minimizer classes address optimization and deterministic calibration and are grouped into:

- **Optimization**: Optimizer provides a base class for the DOTOptimizer, CONMINOptimizer, NPSOLOptimizer, NLPQLPOptimizer, NonlinearCGOptimizer, and SNLLOptimizer gradient-based optimization libraries and the APPSOptimizer (supported by APPSEvalMgr for function evaluations), COLINOptimizer (supported by COLINApplication for function evaluations), JEGAOptimizer, and NCSUOptimizer nongradient-based optimization methods and libraries.

- **Parameter estimation**: LeastSq provides a base class for NL2SOLLeastSq, a least-squares solver based on NL2SOL, SNLLeastSq, 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 SurrogateModel 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 NonDSampling, NonDReliability (reliability analysis), NonDExpansion (stochastic expansion methods), NonDInterval (interval-based epistemic methods), NonDCalibration (nondeterministic calibration), and EfficientSubspaceMethod (prototype input-space dimension reduction method for UQ).
  
  – NonDSampling is further specialized with the NonDLHSSampling class for Latin hypercube and Monte Carlo sampling, the NonDIncremLHSSampling class for incremental Latin hypercube sampling, NonDAdaptImpSampling for multimodal adaptive importance sampling, and NonDGPImpSampling for Gaussian process-based importance sampling.
  
  – NonDReliability is further specialized with local and global methods (NonDLocalReliability and NonDGlobalReliability).
  
  – 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).
  
  – NonDCalibration provides a base class for nondeterministic calibration methods with specialization to Bayesian calibration in NonDBayesCalibration. Specific Bayesian calibration implementations exist in NonDGPMSABayesCalibration and NonDQUESOBayesCalibration.
  
  – NonDInterval provides a base class for epistemic interval-based UQ methods. Three interval analysis approaches are provided: LHS (NonDLHInterval), efficient global optimization (NonDGlobalInterval), and local optimization (NonDLocalInterval). Each of these three has specializations for single interval (NonDLHSSingleInterval, NonDGlobalSingleInterval, NonDLocalSingleInterval) and Dempster-Shafer Theory of Evidence (NonDLHSEvidence, NonDGlobalEvidence, NonDLocalEvidence) approaches.

• Parameter studies and design of experiments: PStudyDACE provides a base class for ParamStudy, which provides capabilities for directed parameter space interrogation, PSUADesignCompExp, which provides access to the Morris One-At-a-Time (MOAT) method for parameter screening, and DDAEDesignCompExp 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 RichExtrapVerification (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.

- **SingleModel**: variables are mapped into responses using a single 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., multiobjective 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 non-blocking 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) TestDriverInterface, 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 are provided 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 below.

- **Multilevel parallel computing:** Dakota supports multiple levels of nested parallelism. A strategy 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 `Strategy`, `ApplicationInterface`, and `ForkApplicationInterface`.

- **Parsing:** Dakota employs the NIDR parser (New Input Deck Reader) to retrieve information from user input files. Parsing options are processed in `CommandLineHandler` and parsing occurs in `ProblemDescDB::manage_inputs()` called from `main.cpp`. NIDR uses the keyword handlers in the NIDR-`ProblemDescDB` derived class to populate data within the `ProblemDescDB` base class, which maintains a `DataStrategy` 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 processed in `CommandLineHandler` and retrieved in `ParallelLibrary::specify_outputs_restart()`, restart file management occurs in `ParallelLibrary::manage_outputs_restart()`, 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: `Strategy`, `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 (`DataStrategy`, `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`.
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.
- 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/developer/ includes information on getting started as a developer and links to project management resources.
- Project web pages are maintained at 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 type-defs 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

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 JAVA-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.
- Feature toggling: when possible, use the "HAVE_<pkg/feature>" convention already in use by many CMake-enabled TPLs, 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)
```

```bash
$ 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 Modify dakota.input.nspec

The master input specification dakota.input.nspec in Dakota/src is the primary file to update when making a specification change. It uses the following syntactic elements:

- () for required group specifications
- [] for optional specifications
- | for alternatives
- {} for functions to process keywords to express logical relationships. These syntactic elements can be used to express various dependency relationships in the input specification. It is recommended that you review the existing specification and have an understanding of the constructs in use before attempting to add new ones.

Warning:

- Do not skip this step. Attempts to modify the NIDR_keywds.hpp file in Dakota/src without using the NIDR table generator are very error-prone. Moreover, the input specification provides a reference to the allowable inputs of a particular executable and should be kept in synch with the parser files; modifying the parser files independent of the input specification creates, at a minimum, undocumented features.
- All keywords in dakota.input.nspec are lower case by convention. All user inputs are converted to lower case by the parser prior to keyword match testing, resulting in case insensitive parsing.
• Since the NIDR parser allows abbreviation of keywords, you must avoid adding a keyword that could be misinterpreted as an abbreviation for a different keyword within the same top-level keyword, such as "strategy" 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 specification.

• 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. With the earlier IDR parser, non-exclusive specifications (those not in mutually exclusive blocks) were required to be unique. That is why there are such aliases for initial_point as cdv_initial_point and ddv_initial_point: so older input files can be used with no or fewer changes.

3.2 Rebuild generated files

When configured with -DENABLE_SPEC_MAINT, performing a make in Dakota/src will regenerate all files which derive from dakota.input.nspec, including NIDR_keywds.hpp, dakota.input.summary, NIDR-guikeywds.h, and dakota.input.desc. 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.

3.3 Update NIDRProblemDescDB.cpp in Dakota/src

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, INTEGERLIST, 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 "strategy" 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 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

```cpp
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

```cpp
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:

```cpp
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

```cpp
void NIDRProblemDescDB::
method_stop(const char *keyname, Values *val, void **g, void *v)
{ 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 ProblemDescDB.cpp in Dakota/src

3.4.1 Augment/update get_\langle data_type \rangle() functions

The next update step involves extending the database retrieval functions in 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,

```c++
...
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 strategy specification is currently allowed so a list of DataStrategy objects is not needed. Rather, ProblemDescDB::strategySpec is the lone DataStrategy object.

To augment the get_\langle data_type \rangle() 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_drv() as:

```c++
{"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").
3.5 Update Corresponding Data Classes

In this step, we extend the Data class definitions (DataStrategy, DataMethod, DataModel, DataVariables, DataInterface, and/or DataResponses) to include the new attributes referenced in Update NIDRProblemDescDB.cpp in Dakota/src and Augment/update get_<data_type>() functions.

3.5.1 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).

3.5.2 Update the .cpp file

Define defaults for the new attributes in the constructor initialization list. Add the new attributes to the assign() function for use by the copy constructor and assignment operator. Add the new attributes to the write(MPIPackBuffer&), read(MPIUnpackBuffer&), and write(ostream&) functions, paying careful attention to the use of a consistent ordering.

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.input.nspec that have been modified should be updated. In particular, the reference manual tables summarizing keywords provide help data to the Jaguar user interface so need to be kept updated.
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

It is possible to link the Dakota toolkit into another application for use as an algorithm library. This section describes facilities which permit this type of integration.

When compiling Dakota with CMake, files in Dakota/src (excepting the main.cpp, restart_util.cpp, and library_mode.cpp main programs) are compiled into libraries that get installed to CMAKE_INSTALL_PREFIX/lib. C/C++ code is in the library dakota_src, while Fortran code lives in the dakota_src_fortran library. Applications may link against these Dakota libraries by specifying appropriate include and link directives. Depending on the configuration used when building this library, other libraries for the vendor optimizers and vendor packages will also be needed to resolve Dakota symbols for DOT, NPSOL, OPT++, NC-SUOpt, LHS, Teuchos, etc. Copies of these libraries are also placed in Dakota/lib. Refer to Linking against the Dakota library for additional information.

Warning:

Users may interface to Dakota as a library within other software applications provided that they abide by the terms of the GNU Lesser General Public License (LGPL). Refer to http://www.gnu.org/licenses/lgpl.html or contact the Dakota team for additional information.

Attention:

The use of Dakota as an algorithm library should be distinguished from the linking of simulations within Dakota using the direct application interface (see DirectApplicInterface). In the former, Dakota is providing algorithm services to another software application, and in the latter, a linked simulation is providing analysis services to Dakota. It is not uncommon for these two capabilities to be used in combination, where a simulation framework provides both the "front end" and the "back end" for Dakota.

5.2 Quick start: examples and test code

To learn by example, refer to the files PluginSerialDirectApplicInterface.[CH] and PluginParallelDirectApplicInterface.[CH] in Dakota/src for simple examples of serial and parallel plug-in
interfaces. The file library_mode.cpp in Dakota/src provides example usage of these plug-ins within a mock
simulator program that demonstrates the required object instantiation syntax in combination with the three
problem database population approaches (input file parsing, data node insertion, and mixed mode). All of this
code may be compiled and tested by configuring Dakota using the --with-plugin option.

5.3 Comparison to main.cpp

The procedure for utilizing Dakota as a library within another application involves a number of steps that
are similar to those used in the stand-alone Dakota application. The stand-alone procedure can be viewed
in the file main.cpp, and the differences for the library approach are most easily explained with reference to
that file. The basic steps of executing Dakota include instantiating the ParallelLibrary, CommandLineHandler,
and ProblemDescDB objects; managing the Dakota input file (ProblemDescDB::manage_inputs()); specifying
restart files and output streams (ParallelLibrary::specify_outputs_restart()); and instantiating the Strategy and
running it (Strategy::run_strategy()). When using Dakota as an algorithm library, the operations are quite sim-
ilar, although command line information (argc, argv, and therefore CommandLineHandler) will not in general
be accessible. In particular, main.cpp can pass argc and argv into the ParallelLibrary and CommandLineHan-
dler constructors and then pass the CommandLineHandler object into ProblemDescDB::manage_inputs() and
ParallelLibrary::specify_outputs_restart(). In an algorithm library approach, a CommandLineHandler object is
not instantiated and overloaded forms of the ParallelLibrary constructor, ProblemDescDB::manage_inputs(), and
ParallelLibrary::specify_outputs_restart() are used.

The overloaded forms of these functions are as follows. For instantiation of the ParallelLibrary object, the default
constructor may be used. This constructor assumes that MPI is administered by the parent application such that
the MPI configuration will be detected rather than explicitly created (i.e., Dakota will not call MPI_Init or MPI_-Finalize). In code, the instantiation

```
ParallelLibrary parallel_lib(argc, argv);
```

is replaced with

```
ParallelLibrary parallel_lib;
```

In the case of specifying restart files and output streams, the call to

```
parallel_lib.specify_outputs_restart(cmd_line_handler);
```

should be replaced with its overloaded form in order to pass the required information through the parameter list

```
parallel_lib.specify_outputs_restart(std_output_filename, std_error_filename,
    read_restart_filename, write_restart_filename, stop_restart_evals);
```

where file names for standard output and error and restart read and write as well as the integer number of restart
evaluations are passed through the parameter list rather than read from the command line of the main Dakota
program. The definition of these attributes is performed elsewhere in the parent application (e.g., specified in the
parent application input file or GUI). In this function call, specify NULL for any files not in use, which will elicit
the desired subset of the following defaults: standard output and standard error are directed to the terminal, no
restart input, and restart output to file dakota.rst. The stop_restart_evals specification is an optional
parameter with a default of 0, which indicates that restart processing should process all records. If no overrides of
these defaults are intended, the call to specify_outputs_restart() may be omitted entirely.

With respect to alternate forms of ProblemDescDB::manage_inputs(), the following section describes different
approaches to populating data within Dakota’s problem description database. It is this database from which all
Dakota objects draw data upon instantiation.
5.4 Problem database population

Now that the ProblemDescDB object has been instantiated, we must populate it with data, either via parsing an input file, direct data insertion, or a mixed approach, as described in the following sections.

5.4.1 Input file parsing

The simplest approach to linking an application with the Dakota library is to rely on Dakota’s normal parsing system to populate Dakota’s problem database (ProblemDescDB) through the reading of an input file. The disadvantage to this approach is the requirement for an additional input file beyond those already required by the parent application.

In this approach, the main.cpp call to

```cpp
problem_db.manage_inputs(cmd_line_handler);
```

would be replaced with its overloaded form

```cpp
problem_db.manage_inputs(dakota_input_file);
```

where the file name for the Dakota input is passed through the parameter list rather than read from the command line of the main Dakota program. Again, the definition of the Dakota input file name is performed elsewhere in the parent application (e.g., specified in the parent application input file or GUI). Refer to run_dakota_parse() in library_mode.cpp for a complete example listing.

ProblemDescDB::manage_inputs() invokes ProblemDescDB::parse_inputs() (which in turn invokes ProblemDescDB::check_input(), ProblemDescDB::broadcast(), and ProblemDescDB::post_process()), which are lower level functions that will be important in the following two sections. Thus, the input file parsing approach may employ a single coarse grain function to coordinate all aspects of problem database population, whereas the two approaches to follow will use lower level functions to accomplish a finer grain of control.

5.4.2 Data node insertion

This approach is more involved than the previous approach, but it allows the application to publish all needed data to Dakota’s database directly, thereby eliminating the need for the parsing of a separate Dakota input file. In this case, ProblemDescDB::manage_inputs() is not called. Rather, DataStrategy, 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 ProblemDescDB::insert_node(), e.g.:

```cpp
// instantiate the data object
DataMethod data_method;

// set the attributes within the data object
data_method.methodName = "nond_sampling";
...

// publish the data object to the ProblemDescDB
problem_db.insert_node(data_method);
```

The data objects are populated with their default values upon instantiation, so only the non-default values need to be specified. Refer to the DataStrategy, DataMethod, DataModel, DataVariables, DataInterface, and DataResponses class documentation and source code for lists of attributes and their defaults.
The default strategy is `single_method`, which runs a single iterator on a single model, and the default model is `single`, so it is not necessary to instantiate and publish a `DataStrategy` or `DataModel` object if advanced multi-component capabilities are not required. Rather, instantiation and insertion of a single `DataMethod`, `DataVariables`, `DataInterface`, and `DataResponses` object is sufficient for basic Dakota capabilities.

Once the data objects have been published to the `ProblemDescDB` object, calls to

```cpp
problem_db.check_input();
problem_db.broadcast();
problem_db.post_process();
```

will perform basic database error checking, broadcast a packed MPI buffer of the specification data to other processors, and post-process specification data to fill in vector defaults (scalar defaults are handled in the Data class constructors), respectively. For parallel applications, processor rank 0 should be responsible for Data node population and insertion and the call to `ProblemDescDB::check_input()`, and all processors should participate in `ProblemDescDB::broadcast()` and `ProblemDescDB::post_process()`. Moreover, preserving the order shown assures that large default vectors are not transmitted by MPI. Refer to `run_dakota_data()` in `library_mode.cpp` for a complete example listing.

### 5.4.3 Mixed mode

In this case, we will combine the parsing of a Dakota input file with some direct database updates. The motivation for this approach arises in large-scale applications where large vectors can be awkward to specify in a Dakota input file. The first step is to parse the input file, but rather than using

```cpp
problem_db.manage_inputs(dakota_input_file);
```

as described in Input file parsing, we will use the lower level function

```cpp
problem_db.parse_inputs(dakota_input_file);
```

to provide a finer grain of control. The passed input file `dakota_input_file` must contain all required inputs. 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. Only the variable/response counts, e.g.:

```plaintext
method
  linear_inequality_constraints = 500

variables
  continuous_design = 1000

responses
  objective_functions = 1
  nonlinear_inequality_constraints = 100000
```

are required in this case. To update the data omissions from their defaults, one uses the `ProblemDescDB::set()` family of overloaded functions, e.g.

```cpp
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem_db.set("variables.continuous_design.initial_point", drv);
```
where the string identifiers are the same identifiers used when pulling information from the database using one of the get_<datatype>() functions (refer to the source code of ProblemDescDB.cpp 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.

If performing these updates within the constructor of a DirectApplicInterface extension/derivation (see Defining the direct application 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 strategy/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 operations at a higher level (e.g., within your main program), prior to Strategy instantiation and execution, 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.:

```
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:

```
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 hardwiring specification identifiers is undesirable, then

```
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. In each of these cases, setting list nodes unlocks the corresponding portions of the database, allowing set/get operations.

Once all direct database updates have been performed in this manner, calls to ProblemDescDB::broadcast() and ProblemDescDB::post_process() should be used on all processors. The former will broadcast a packed MPI buffer with the aggregated set of specification data from rank 0 to other processors, and the latter will 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). Refer to run_dakota_mixed() in library_mode.cpp for a complete example listing.

### 5.5 Instantiating the strategy

With the ProblemDescDB object populated with problem data, we may now instantiate the strategy.

```
// instantiate the strategy
Strategy selected_strategy(problem_db);
```

Following strategy 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:
// 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, where the number of MPI
communicators in the array corresponds to one communicator per ParallelConfiguration instance.

5.6 Defining the direct application interface

When employing a library interface to Dakota, it is frequently desirable to also use a direct interface between
Dakota and the simulation. There are two approaches to defining this direct interface.

5.6.1 Extension

The first approach involves extending the existing DirectApplicInterface class to support additional di-
rect simulation interfaces. In this case, a new simulation interface function can be added to
Dakota/src/DirectApplicInterface.[CH] for the simulation of interest. If the new function will not be a member
function, then the following prototype should be used in order to pass the required data:

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

If the new function will be a member function, then this can be simplified to

int sim();

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

This simulation can then be added to the logic blocks in DirectApplicInterface::derived_map_ac(). In addition,
DirectApplicInterface::derived_map_if() and DirectApplicInterface::derived_map_of() can be extended to per-
form 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 may 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 following derivation approach should be employed.

5.6.2 Derivation

The second approach is to derive a new interface from DirectApplicInterface in order to redefine several virtual
functions. A typical derived class declaration might be

namespace SIM {

class SerialDirectApplicInterface: public Dakota::DirectApplicInterface
{
public:

    // Constructor and destructor
    SerialDirectApplicInterface(const Dakota::ProblemDescDB& problem_db);
    ~SerialDirectApplicInterface();
}
5.6 Defining the direct application interface

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.

The new derived interface object (from namespace SIM) must now be plugged into the strategy. In the simplest case of a single model and interface, one could use

```
// retrieve the interface of interest
ModelList& all_models = problem_db.model_list();
Model& first_model = *all_models.begin();
Interface& interface = first_model.interface();
// plug in the new direct interface instance (DB does not need to be set)
interface.assign_rep(new SIM::SerialDirectApplicInterface(problem_db), false);
```

definitions from within the Dakota namespace. In a more advanced case of multiple models and multiple interface plug-ins, one might use

```
// retrieve the list of Models from the Strategy
ModelList& models = problem_db.model_list();
// iterate over the Model list
for (ModelIterator ml_iter = models.begin(); ml_iter != models.end(); ml_iter++) {

    Interface& interface = ml_iter->interface();
    if (interface.interface_type() == "direct" &&
        interface.analysis_drivers().contains("SIM") ) {
        // set the correct list nodes within the DB prior to new instantiations
        problem_db.set_db_model_nodes(ml_iter->model_id());
        // plug in the new direct interface instance
        interface.assign_rep(new SIM::SerialDirectApplicInterface(problem_db), false);
    }
}
```

In the 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

```
const ParallelLevel& ea_level =
    ml_iter->parallel_configuration_iterator()->ea_parallel_level();
const MPI_Comm& analysis_comm = ea_level.server_intra_communicator();
interface.assign_rep(new SIM::ParallelDirectApplicInterface(problem_db, analysis_comm), false);
```
Since Models 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 in Instantiating the strategy), 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 runtime (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.

### 5.7 Additional updates

As part of strategy instantiation, all problem specification data is extracted from ProblemDescDB as various objects are constructed. Therefore, any updates that need to be performed following strategy instantiation must be performed through direct set operations on the constructed objects. In the previous section, the process for updating the Interface object used within a Model was shown. 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);
```

### 5.8 Executing the strategy

Finally, with simulation configuration and plug-ins completed, we execute the strategy:

```cpp
// run the strategy
selected_strategy.run_strategy();
```

### 5.9 Retrieving data after a run

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

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

// retrieve the final response values
const Response& resp = selected_strategy.response_results();
```
5.10 Linking against the Dakota library

In the case of optimization, the final design is returned, and in the case of uncertainty quantification, the final statistics are returned.

5.10 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 install to CMAKE_INSTALL_PREFIX/bin and CMAKE_INSTALL_PREFIX/lib. When running CMake, Dakota and Dakota-included third-party libraries will be output as TPL LIBS, e.g.,

-- TPL LIBS: nidr;teuchos;pecos;pecos_src;lhs;mods;mod;fftpack;sparsegrid;
    surfpack;surfpack;surfpack_fortran;utilib;colin;interfaces;scollib;3po;pebbl;
    tinyxml;commin;dace;analyzer;random;sampling;bose;dot;fsu;hoppack;jega;
    jega_fe;mosa;soa;eutils;utilities;ncsuopt;nlpql;copport;nomad;npsol;optpp;psuade;
    DGraphics;amplsolver

While external dependencies will be output as EXTRA TPL LIBS:

-- EXTRA TPL LIBS: /usr/lib64/openmpi/lib/libmpi_cxx.so;optimized;
    /usr/lib64/libboost_regex-mt.so;debug;/usr/lib64/libboost_regex-mt.so;
    optimized;/usr/lib64/libboost_filesystem-mt.so;debug;
    /usr/lib64/libboost_filesystem-mt.so;optimized;/usr/lib64/libboost_system-mt.so;
    debug;/usr/lib64/libboost_system-mt.so;optimized;
    /usr/lib64/libSM.so;/usr/lib64/libICE.so;/usr/lib64/libX11.so;
    /usr/lib64/libXext.so;/usr/lib64/libXm.so;/usr/lib64/libXpm.so;
    /usr/lib64/libXmu.so;/usr/lib64/libXt.so;-lpthread;/usr/lib64/liblapack.so;
    /usr/lib64/libblas.so

Note that depending on how you configured Dakota, some of the libraries may not be included (for example NPSOL, DOT, NLPQL). Optional libraries like GSL (discouraged due to GPL license) may also be needed if Dakota was configured with them. Check which appear in CMAKE_INSTALL_PREFIX/bin CMAKE_INSTALL_PREFIX/lib.

Note that as of Dakota 5.2, -lnewmat is no longer required but additional Boost libraries are needed (-lboost_regex -lboost_filesystem -lboost_system) as a result of migration from legacy Dakota utilities to more modern Boost components.

You may also need funcadd0.o, -lf1, -lexpat, and, if linking with system-provided GSL, -lgslcblas. The AMPL solver library may require -ldl. System compiler and math libraries may also need to be included. If configuring with graphics, you will need to add -lDGraphics and system X libraries (partial list here):

-lXpm -lXm -lXt -lXmu -lXp -lXext -lX11 -lSM -lICE

We have experienced problems with the creation of libamplsolver.a on some platforms. Please use the Dakota mailing lists for help with any problems.

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. This ensures that the platform configuration settings are properly propagated.
5.11 Summary

To utilize the Dakota library within a parent software application, the basic steps of main.cpp and the order of invocation of these steps should be mimicked from within the parent application. Of these steps, ParallelLibrary instantiation, ProblemDescDB::manage_inputs() and ParallelLibrary::specify_outputs_restart() require the use of overloaded forms in order to function in an environment without direct command line access and, potentially, without file parsing. Additional optional steps not performed in main.cpp include the extension/derivation of the direct interface and the retrieval of strategy results after a run.

Dakota’s library mode is now in production use within several Sandia and external simulation codes/frameworks.
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::compute_response() 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::compute_response() utilizes Model::derived_compute_response() for portions of the response computation specific to derived model classes.
- Model::derived_compute_response() 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::asynch_compute_response() 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::asynch_compute_response() utilizes Model::derived_asynch_compute_response() 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.

• 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:
  - asynchronous local mode (using ApplicationInterface::asynchronous_local_evaluations() or ApplicationInterface::asynchronous_local_evaluations_nowait())
  - message passing mode (using ApplicationInterface::self_schedule_evaluations() or ApplicationInterface::static_schedule_evaluations() on the iterator master and ApplicationInterface::serve_evaluations_synch() or ApplicationInterface::serve_evaluations_asynch_peer() on the servers)
  - hybrid mode (using ApplicationInterface::self_schedule_evaluations() or ApplicationInterface::static_schedule_evaluations() on the iterator master and ApplicationInterface::serve_evaluations_asynch() or ApplicationInterface::serve_evaluations_asynch_peer() on the servers)

• These scheduling functions utilize ApplicationInterface::derived_map() and ApplicationInterface::derived_map_asynch() for portions of asynchronous job launching specific to derived application interface classes, as well as ApplicationInterface::derived_synch() and ApplicationInterface::derived_synch_nowait() for portions of job capturing specific to derived application interface classes.

6.3 Analyses within each function evaluation

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 (ForkApplicInterface::synchronous_local_analyses(), ForkApplicInterface::asynchronous_local_analyses(), ApplicationInterface::self_schedule_analyses(), ApplicationInterface::serve_analyses_synch(), ForkApplicInterface::serve_analyses_asynch()) 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
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 allDiscreteIntVars 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.
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.

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 MixedVariables 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.
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) ...................... 65
- SIM (A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA) .... 240
Chapter 9

Class Index

9.1 Class Hierarchy

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

- ActiveSet ........................................... 241
- Approximation ........................................ 269
  - GaussProcApproximation ......................... 462
  - PecosApproximation ................................ 804
  - SurfpackApproximation ............................ 944
  - TANA3Approximation ............................... 976
  - TaylorApproximation ............................. 979
- APPSEvalMgr ........................................ 286
- BaseConstructor .................................... 293
- BiStream ............................................. 294
- BoStream ............................................. 297
- COLINAApplication ................................. 300
- CommandShell ....................................... 314
- Constraints .......................................... 327
  - MixedVarConstraints ............................... 542
  - RelaxedVarConstraints ............................ 856
- Interface .............................................. 354
- DataMethod .......................................... 356
- DataMethodRep ..................................... 358
- DataModel ............................................ 374
  - DataModelRep ...................................... 376
- DataResponses ....................................... 382
- DataResponsesRep ................................... 384
- DataStrategy ........................................ 389
  - DataStrategyRep ................................... 391
- DataVariables ....................................... 394
  - DataVariablesRep .................................. 396
- DiscrepancyCorrection ............................. 420
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td>430</td>
</tr>
<tr>
<td>Evaluator</td>
<td>442</td>
</tr>
<tr>
<td>Evaluator</td>
<td>444</td>
</tr>
<tr>
<td>EvaluatorCreator</td>
<td>451</td>
</tr>
<tr>
<td>ExperimentData</td>
<td>453</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>469</td>
</tr>
<tr>
<td>CommandLineHandler</td>
<td>311</td>
</tr>
<tr>
<td>Graphics</td>
<td>473</td>
</tr>
<tr>
<td>Interface</td>
<td>487</td>
</tr>
<tr>
<td>ApplicationInterface</td>
<td>251</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
<td>414</td>
</tr>
<tr>
<td>MatlabInterface</td>
<td>526</td>
</tr>
<tr>
<td>PythonInterface</td>
<td>841</td>
</tr>
<tr>
<td>ScilabInterface</td>
<td>889</td>
</tr>
<tr>
<td>TestDriverInterface</td>
<td>981</td>
</tr>
<tr>
<td>ParallelDirectApplicInterface</td>
<td>771</td>
</tr>
<tr>
<td>SerialDirectApplicInterface</td>
<td>898</td>
</tr>
<tr>
<td>ProcessApplicInterface</td>
<td>824</td>
</tr>
<tr>
<td>ProcessHandleApplicInterface</td>
<td>829</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>454</td>
</tr>
<tr>
<td>SpawnApplicInterface</td>
<td>933</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
<td>971</td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>477</td>
</tr>
<tr>
<td>ApproximationInterface</td>
<td>279</td>
</tr>
<tr>
<td>Iterator</td>
<td>498</td>
</tr>
<tr>
<td>Analyzer</td>
<td>245</td>
</tr>
<tr>
<td>NonD</td>
<td>622</td>
</tr>
<tr>
<td>EfficientSubspaceMethod</td>
<td>435</td>
</tr>
<tr>
<td>NonDCalibration</td>
<td>646</td>
</tr>
<tr>
<td>NonDBayesCalibration</td>
<td>644</td>
</tr>
<tr>
<td>NonDDREAMBayesCalibration</td>
<td>652</td>
</tr>
<tr>
<td>NonDGPSABayesCalibration</td>
<td>680</td>
</tr>
<tr>
<td>NonDQUESOBayesCalibration</td>
<td>735</td>
</tr>
<tr>
<td>NonDExpansion</td>
<td>657</td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>725</td>
</tr>
<tr>
<td>NonDStochCollocation</td>
<td>752</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>687</td>
</tr>
<tr>
<td>NonDCubature</td>
<td>649</td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>730</td>
</tr>
<tr>
<td>NonDSparseGrid</td>
<td>748</td>
</tr>
<tr>
<td>NonDInterval</td>
<td>691</td>
</tr>
<tr>
<td>NonDGlobalInterval</td>
<td>667</td>
</tr>
<tr>
<td>NonDGlobalEvidence</td>
<td>665</td>
</tr>
<tr>
<td>NonDGlobalSingleInterval</td>
<td>674</td>
</tr>
<tr>
<td>NonDLSHInterval</td>
<td>696</td>
</tr>
<tr>
<td>NonDLSHEvidence</td>
<td>694</td>
</tr>
<tr>
<td>NonDLSHSSingleInterval</td>
<td>701</td>
</tr>
</tbody>
</table>
## 9.1 Class Hierarchy

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonDLocalInterval</td>
<td>705</td>
</tr>
<tr>
<td>NonDLocalEvidence</td>
<td>703</td>
</tr>
<tr>
<td>NonDLocalSingleInterval</td>
<td>720</td>
</tr>
<tr>
<td>NonDPOFDarts</td>
<td>722</td>
</tr>
<tr>
<td>NonDReliability</td>
<td>738</td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
<td>671</td>
</tr>
<tr>
<td>NonDLocalReliability</td>
<td>708</td>
</tr>
<tr>
<td>NonDSampling</td>
<td>740</td>
</tr>
<tr>
<td>NonDAadaptImpSampling</td>
<td>634</td>
</tr>
<tr>
<td>NonDAadaptiveSampling</td>
<td>638</td>
</tr>
<tr>
<td>NonDGPImpSampling</td>
<td>676</td>
</tr>
<tr>
<td>NonDIncremLHSSampling</td>
<td>684</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>698</td>
</tr>
<tr>
<td>PStudyDACE</td>
<td>834</td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>409</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>457</td>
</tr>
<tr>
<td>ParamStudy</td>
<td>796</td>
</tr>
<tr>
<td>PSUADEDesignCompExp</td>
<td>837</td>
</tr>
<tr>
<td>Verification</td>
<td>1003</td>
</tr>
<tr>
<td>RichExtrapVerification</td>
<td>886</td>
</tr>
<tr>
<td>Minimizer</td>
<td>528</td>
</tr>
<tr>
<td>LeastSq</td>
<td>521</td>
</tr>
<tr>
<td>NL2SOLLeastSq</td>
<td>607</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>616</td>
</tr>
<tr>
<td>SNLLLLeastSq</td>
<td>917</td>
</tr>
<tr>
<td>Optimizer</td>
<td>763</td>
</tr>
<tr>
<td>APPSOptimizer</td>
<td>289</td>
</tr>
<tr>
<td>COLINOptimizer</td>
<td>304</td>
</tr>
<tr>
<td>CONMINOptimizer</td>
<td>319</td>
</tr>
<tr>
<td>DOTOptimizer</td>
<td>425</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>513</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>587</td>
</tr>
<tr>
<td>NLPQLPOptimizer</td>
<td>610</td>
</tr>
<tr>
<td>NomadOptimizer</td>
<td>619</td>
</tr>
<tr>
<td>NonlinearCGOptimizer</td>
<td>755</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>759</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>922</td>
</tr>
<tr>
<td>SurrBasedMinimizer</td>
<td>960</td>
</tr>
<tr>
<td>EffGlobalMinimizer</td>
<td>432</td>
</tr>
<tr>
<td>SurrBasedGlobalMinimizer</td>
<td>949</td>
</tr>
<tr>
<td>SurrBasedLocalMinimizer</td>
<td>951</td>
</tr>
<tr>
<td>Model</td>
<td>546</td>
</tr>
<tr>
<td>NestedModel</td>
<td>591</td>
</tr>
<tr>
<td>RecastModel</td>
<td>845</td>
</tr>
<tr>
<td>SingleModel</td>
<td>911</td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>966</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
<td>340</td>
</tr>
<tr>
<td>HierarchSurrModel</td>
<td>480</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>581</td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>584</td>
</tr>
<tr>
<td>NL2Res</td>
<td>606</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>618</td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>769</td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>773</td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>777</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>792</td>
</tr>
<tr>
<td>partial_prp_equality</td>
<td>802</td>
</tr>
<tr>
<td>partial_prp_hash</td>
<td>803</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>812</td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
<td>601</td>
</tr>
<tr>
<td>RecastBaseConstructor</td>
<td>844</td>
</tr>
<tr>
<td>Response</td>
<td>860</td>
</tr>
<tr>
<td>ResponseRep</td>
<td>866</td>
</tr>
<tr>
<td>ResultsDBAny</td>
<td>874</td>
</tr>
<tr>
<td>ResultsEntry&lt;StoredType&gt;</td>
<td>877</td>
</tr>
<tr>
<td>ResultsID</td>
<td>879</td>
</tr>
<tr>
<td>ResultsManager</td>
<td>881</td>
</tr>
<tr>
<td>ResultsNames</td>
<td>884</td>
</tr>
<tr>
<td>SensAnalysisGlobal</td>
<td>891</td>
</tr>
<tr>
<td>SharedVariablesData</td>
<td>900</td>
</tr>
<tr>
<td>SharedVariablesDataRep</td>
<td>905</td>
</tr>
<tr>
<td>SNLLBase</td>
<td>914</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>917</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>922</td>
</tr>
<tr>
<td>SOLBase</td>
<td>930</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>616</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>759</td>
</tr>
<tr>
<td>Strategy</td>
<td>935</td>
</tr>
<tr>
<td>ConcurrentStrategy</td>
<td>316</td>
</tr>
<tr>
<td>HybridStrategy</td>
<td>485</td>
</tr>
<tr>
<td>CollaborativeHybridStrategy</td>
<td>309</td>
</tr>
<tr>
<td>EmbeddedHybridStrategy</td>
<td>440</td>
</tr>
<tr>
<td>SequentialHybridStrategy</td>
<td>893</td>
</tr>
<tr>
<td>SingleMethodStrategy</td>
<td>909</td>
</tr>
<tr>
<td>TrackerHTTP</td>
<td>987</td>
</tr>
<tr>
<td>Variables</td>
<td>990</td>
</tr>
<tr>
<td>MixedVariables</td>
<td>544</td>
</tr>
<tr>
<td>RelaxedVariables</td>
<td>858</td>
</tr>
</tbody>
</table>
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) ................................................... 241
- **Analyzer** (Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy) .......... 245
- **ApplicationInterface** (Derived class within the interface class hierarchy for supporting interfaces to simulation codes) .......................................................... 251
- **Approximation** (Base class for the approximation class hierarchy) ................................................. 269
- **ApproximationInterface** (Derived class within the interface class hierarchy for supporting approximations to simulation-based results) ................................... 279
- **APPSEvalMgr** (Evaluation manager class for APPSPACK) .......................................................... 286
- **APPSOptimizer** (Wrapper class for APPSPACK) ................................................................. 289
- **BaseConstructor** (Dummy struct for overloading letter-envelope constructors) ............................. 293
- **BiStream** (The binary input stream class. Overloads the $\gg$ operator for all data types) ............... 294
- **BoStream** (The binary output stream class. Overloads the $\ll$ operator for all data types) ............. 297
- **COLINApplication** .................................................................................................................. 300
- **COLINOptimizer** (Wrapper class for optimizers defined using COLIN) .................................. 304
- **CollaborativeHybridStrategy** (**Strategy** for hybrid minimization using multiple collaborating optimization and nonlinear least squares methods) ................................. 309
- **CommandLineHandler** (Utility class for managing command line inputs to DAKOTA) .............. 311
- **CommandShell** (Utility class which defines convenience operators for spawning processes with system calls) ...................................................................................... 314
- **ConcurrentStrategy** (**Strategy** for multi-start iteration or pareto set optimization) ................. 316
- **CONMINOptimizer** (Wrapper class for the CONMIN optimization library) ............................. 319
- **Constraints** (Base class for the variable constraints class hierarchy) .......................................... 327
- **DataFitSurrModel** (Derived model class within the surrogate model branch for managing data fit surrogates (global and local)) .................................................. 340
- **DataInterface** (Handle class for interface specification data) .................................................... 354
- **DataMethod** (Handle class for method specification data) ......................................................... 356
- **DataMethodRep** (Body class for method specification data) ..................................................... 358
- **DataModel** (Handle class for model specification data) ............................................................ 374
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataModelRep</td>
<td>(Body class for model specification data)</td>
<td>376</td>
</tr>
<tr>
<td>DataResponses</td>
<td>(Handle class for responses specification data)</td>
<td>382</td>
</tr>
<tr>
<td>DataResponsesRep</td>
<td>(Body class for responses specification data)</td>
<td>384</td>
</tr>
<tr>
<td>DataStrategy</td>
<td>(Handle class for strategy specification data)</td>
<td>389</td>
</tr>
<tr>
<td>DataStrategyRep</td>
<td>(Body class for strategy specification data)</td>
<td>391</td>
</tr>
<tr>
<td>DataVariables</td>
<td>(Handle class for variables specification data)</td>
<td>394</td>
</tr>
<tr>
<td>DataVariablesRep</td>
<td>(Body class for variables specification data)</td>
<td>396</td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>(Wrapper class for the DDACE design of experiments library)</td>
<td>409</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
<td>(Derived application interface class which spawns simulation codes and testers</td>
<td>414</td>
</tr>
<tr>
<td>DiscrepancyCorrection</td>
<td>(Base class for discrepancy corrections)</td>
<td>420</td>
</tr>
<tr>
<td>DOTOptimizer</td>
<td>(Wrapper class for the DOT optimization library)</td>
<td>425</td>
</tr>
<tr>
<td>Driver</td>
<td>(A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm)</td>
<td>430</td>
</tr>
<tr>
<td>EffGlobalMinimizer</td>
<td>(Implementation of Efficient Global Optimization/Least Squares algorithms)</td>
<td>432</td>
</tr>
<tr>
<td>EfficientSubspaceMethod</td>
<td>(Efficient Subspace Method (ESM), as proposed by Hany S. Abdel-Khalik)</td>
<td>435</td>
</tr>
<tr>
<td>EmbeddedHybridStrategy</td>
<td>(Strategy for closely-coupled hybrid minimization, typically involving the embedding of local search methods within global search methods)</td>
<td>440</td>
</tr>
<tr>
<td>Evaluator</td>
<td>(NOMAD-based Evaluator class)</td>
<td>442</td>
</tr>
<tr>
<td>Evaluator</td>
<td>(An evaluator specialization that knows how to interact with Dakota)</td>
<td>444</td>
</tr>
<tr>
<td>EvaluatorCreator</td>
<td>(A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator)</td>
<td>451</td>
</tr>
<tr>
<td>ExperimentData</td>
<td></td>
<td>453</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>(Derived application interface class which spawns simulation codes using fork/execvp/waitpid)</td>
<td>454</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>(Wrapper class for the FSUDace QMC/CVT library)</td>
<td>457</td>
</tr>
<tr>
<td>GaussProcApproximation</td>
<td>(Derived approximation class for Gaussian Process implementation)</td>
<td>462</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>(GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France))</td>
<td>469</td>
</tr>
<tr>
<td>Graphics</td>
<td>(Single interface to 2D (motif) and 3D (PLPLOT) graphics as well as tabular cataloguing of data for post-processing with Matlab, Tecplot, etc)</td>
<td>473</td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>(Derived application interface class which spawns simulation codes using grid services such as Condor or Globus)</td>
<td>477</td>
</tr>
<tr>
<td>HierarchSurrModel</td>
<td>(Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity))</td>
<td>480</td>
</tr>
<tr>
<td>HybridStrategy</td>
<td>(Base class for hybrid minimization strategies)</td>
<td>485</td>
</tr>
<tr>
<td>Interface</td>
<td>(Base class for the interface class hierarchy)</td>
<td>487</td>
</tr>
<tr>
<td>Iterator</td>
<td>(Base class for the iterator class hierarchy)</td>
<td>498</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>(A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA))</td>
<td>513</td>
</tr>
<tr>
<td>LeastSq</td>
<td>(Base class for the nonlinear least squares branch of the iterator hierarchy)</td>
<td>521</td>
</tr>
<tr>
<td>MatlabInterface</td>
<td></td>
<td>526</td>
</tr>
<tr>
<td>Minimizer</td>
<td>(Base class for the optimizer and least squares branches of the iterator hierarchy)</td>
<td>528</td>
</tr>
<tr>
<td>MixedVarConstraints</td>
<td>(Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging))</td>
<td>542</td>
</tr>
<tr>
<td>MixedVariables</td>
<td>(Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging))</td>
<td>544</td>
</tr>
<tr>
<td>Model</td>
<td>(Base class for the model class hierarchy)</td>
<td>546</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>(Class for packing MPI message buffers)</td>
<td>581</td>
</tr>
</tbody>
</table>
MPIUnpackBuffer (Class for unpacking MPI message buffers) ........................................ 584
NCSUOptimizer (Wrapper class for the NCSU DIRECT optimization library) .......................... 587
NestedModel (Derived model class which performs a complete sub-iterator execution within every 
evaluation of the model) ........................................................................................................ 591
NIDRProblemDescDB (The derived input file database utilizing the new IDR parser) ................. 601
NL2Res (Auxiliary information passed to calcr and calcj via ur) ............................................... 606
NL2SOLLeastSq (Wrapper class for the NL2SOL nonlinear least squares library) ................. 607
NLPQLPOptimizer (Wrapper class for the NLPQLP optimization library, Version 2.0) .............. 610
NLSSOLLeastSq (Wrapper class for the NLSSOL nonlinear least squares library) ................... 616
NoDBBaseConstructor (Dummy struct for overloading constructors used in on-the-fly instantiations) 618
NomadOptimizer (Wrapper class for NOMAD Optimizer) .................................................... 619
NonD (Base class for all nondeterministic iterators (the DAKOTA/UQ branch)) ......................... 622
NonDAdaptImpSampling (Class for the Adaptive Importance Sampling methods within DAKOTA) 634
NonDAdaptiveSampling (Class for testing various Adaptively sampling methods using geometric, 
estatistical, and topological information of the surrogate) .................................................... 638
NonDBayesCalibration (Base class for Bayesian inference: generates posterior distribution on model 
parameters given experimental data) ......................................................................................... 644
NonDCalibration ................................................................................................................. 646
NonDCubature (Derived nondeterministic class that generates N-dimensional numerical cubature points 
for evaluation of expectation integrals) ..................................................................................... 649
NonDDREAMBayesCalibration (Bayesian inference using the DREAM approach) ....................... 652
NonDExpansion (Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC)) 657
NonDGGlobalEvidence (Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ) 665
NonDGGlobalInterval (Class for using global non-gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification) ..................... 667
NonDGGlobalReliability (Class for global reliability methods within DAKOTA/UQ) .................. 671
NonDGGlobalSingleInterval (Class for using global non-gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification)......................... 674
NonDGImpSampling (Class for the Gaussian Process-based Importance Sampling method) .......... 676
NonDGPMASBayesCalibration (Generates posterior distribution on model parameters given experimental data) .......................................................................................................................... 680
NonDIncrementLHSSampling (Performs incremental LHS sampling for uncertainty quantification) .. 684
NonDIntegration (Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals) .................................................... 687
NonDInterval (Base class for interval-based methods within DAKOTA/UQ) .............................. 691
NonDLHSEvidence (Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ) .. 694
NonDLHSInterval (Class for the LHS-based interval methods within DAKOTA/UQ) ............... 696
NonDLHSSampling (Performs LHS and Monte Carlo sampling for uncertainty quantification) .... 698
NonDLHSsingleInterval (Class for pure interval propagation using LHS) .................................. 701
NonDLocalEvidence (Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ) 703
NonDLocalInterval (Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification) ........................................ 705
NonDLocalReliability (Class for the reliability methods within DAKOTA/UQ) ......................... 708
NonDLocalSingleInterval (Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification) .......................... 720
NonDPOFDarts (Base class for POF Dart methods within DAKOTA/UQ) ................................... 722
NonDPolynomialChaos (Nonintrusive polynomial chaos expansion approaches to uncertainty quantification) ...................................................................................................................... 725
NonDQuadrature (Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normal/uniforms/exponentials/betas/gammas) ........................................... 730
NonDQUESOBayesCalibration (Bayesian inference using the QUESO library from UT Austin) .......................... 735
NonDReliability (Base class for the reliability methods within DAKOTA/UQ) ..................................................... 738
NonDSampling (Base class for common code between NonDLHSSampling, NonDIncremLHSSampling, and NonDAdaptImpSampling) .......................................................... 740
NonDSparseGrid (Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables) 748
NonDStochCollocation (Nonintrusive stochastic collocation approaches to uncertainty quantification) .............. 752
NonlinearCGOptimizer ................................................................. 755
NPSOLOptimizer (Wrapper class for the NPSOL optimization library) .............................................................. 759
Optimizer (Base class for the optimizer branch of the iterator hierarchy) ......................................................... 763
ParallelConfiguration (Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration) ........................................... 769
ParallelDirectApplicInterface (Sample derived interface class for testing parallel simulator plug-ins using assign_rep() ) ................................................................. 771
ParallelLevel (Container class for the data associated with a single level of communicator partitioning) .......... 773
ParallelLibrary (Class for partitioning multiple levels of parallelism and managing message passing within these levels) ........................................................................................................ 777
ParamResponsePair (Container class for a variables object, a response object, and an evaluation id) .............. 792
ParamStudy (Class for vector, list, centered, and multidimensional parameter studies) .................................. 796
partial_prp_equality (Predicate for comparing ONLY the interfaceId and Vars attributes of PRPair) ............ 802
partial_prp_hash (Wrapper to delegate to the ParamResponsePair hash_value function) ............................... 803
PecosApproximation (Derived approximation class for global basis polynomials) ......................................... 804
ProblemDescDB (The database containing information parsed from the DAKOTA input file) ......................... 812
ProcessApplicInterface (Derived application interface class that spawns a simulation code using a separate process and communicates with it through files) ............................................ 824
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) ........................................................................................................ 829
PStudyDACE (Base class for managing common aspects of parameter studies and design of experiments methods) .............................................................................................................. 834
PSUADEDesignCompExp (Wrapper class for the PSUADE library) ................................................................. 837
PythonInterface ......................................................................... 841
RecastBaseConstructor (Dummy struct for overloading constructors used in on-the-fly Model instantiations) .... 844
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) ...................................................... 845
RelaxedVarConstraints (Derived class within the Constraints hierarchy which employs relaxation of discrete variables) ....................................................................................................... 845
RelaxedVariables (Derived class within the Variables hierarchy which employs the relaxation of discrete variables) ........................................................................................................... 856
Response (Container class for response functions and their derivatives. Response provides the handle class) .......................................................................................................................... 858
ResponseRep (Container class for response functions and their derivatives. ResponseRep provides the body class) ............................................................................................................. 860
ResultsDBAny ............................................................................. 866
ResultsEntry<StoredType> (Class to manage in-core vs. file database lookups) ................................................. 872
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 run_iterator call creates or increments this count for its string identifier ) .................................................. 879
ResultsManager (Results manager for iterator final data ) ................................................................. 881
ResultsNames (List of valid names for iterator results ) ................................................................. 884
RichExtrapVerification (Class for Richardson extrapolation for code and solution verification ) ................. 886
SciLibInterface ................................................................. 889
SensAnalysisGlobal (Class for a utility class containing correlation calculations and variance-based decomposition ) .................................................................................. 891
SequentialHybridStrategy (Strategy for sequential hybrid minimization using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity ) ........................................... 893
SerialDirectApplicInterface (Sample derived interface class for testing serial simulator plug-ins using assign_rep()) .......................................................................................................................... 898
SharedVariablesData (Container class encapsulating variables data that can be shared among a set of Variables instances ) ........................................................................................................ 900
SharedVariablesDataRep (The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances ) ............... 905
SingleMethodStrategy (Simple fall-through strategy for running a single iterator on a single model ) ........ 909
SingleModel (Derived model class which utilizes a single interface to map variables into responses ) .......... 911
SNLLBase (Base class for OPT++ optimization and least squares methods ) ............................................. 914
SNLLLeastSq (Wrapper class for the OPT++ optimization library ) ......................................................... 917
SNLLOptimizer (Wrapper class for the OPT++ optimization library ) .................................................... 922
SOLBase (Base class for Stanford SOL software ) .............................................................................. 930
SpawnApplicInterface (Derived application interface class which spawns simulation codes using spawnvp) .................................................................................................................. 933
Strategy (Base class for the strategy class hierarchy ) ........................................................................... 935
SurpackApproximation (Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota ) .......................................................................... 944
SurrBasedGlobalMinimizer (The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls ) ........................................... 949
SurrBasedLocalMinimizer (Class for provably-convergent local surrogate-based optimization and non-linear least squares ) ................................................................................................. 951
SurrBasedMinimizer (Base class for local/global surrogate-based optimization/least squares ) ............... 960
SurrogateModel (Base class for surrogate models (DataFitSurrModel and HierarchSurrModel) ) ............ 966
SysCallApplicInterface (Derived application interface class which spawns simulation codes using system calls ) ...................................................................................................................... 971
TANA3Approximation (Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation) ) .................................................................................. 976
TaylorApproximation (Derived approximation class for first- or second-order Taylor series (a local approximation) ) .................................................................................................................. 979
TestDriverInterface .......................................................................................................................... 981
TrackerHTTP (TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library ) ........... 987
Variables (Base class for the variables class hierarchy ) ......................................................................... 990
Verification (Base class for managing common aspects of verification studies ) ........................................ 1003
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) ............................................ 1005
- dakota_dll_api.h (API for DLL interactions) ................................................................. 1007
- 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) .................................................. 1009
- JEGAOptimizer.cpp (Contains the implementation of the JEGAOptimizer class) ................. 1011
- JEGAOptimizer.hpp (Contains the definition of the JEGAOptimizer class) .......................... 1012
- library_mode.cpp (File containing a mock simulator main for testing DAKOTA in library mode) .... 1013
- library_split.cpp (File containing a mock simulator main for testing DAKOTA in library mode on a split communicator) ................................................................. 1015
- main.cpp (File containing the main program for DAKOTA) ................................................... 1016
- restart_util.cpp (File containing the DAKOTA restart utility main program) .......................... 1017
Chapter 12

Namespace Documentation

12.1 Dakota Namespace Reference

The primary namespace for DAKOTA.

Classes

- 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 APPSOptimizer
  Wrapper class for APPSPACK.

- class COLINApplication
- class COLINOptimizer
  Wrapper class for optimizers defined using COLIN.

- class CollaborativeHybridStrategy
  Strategy for hybrid minimization 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 ConcurrentStrategy**
  Strategy for multi-start iteration or pareto set optimization.

- **class CONMINOptimizer**
  Wrapper class for the CONMIN optimization library.

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

- **struct NoDBBaseConstructor**
  Dummy struct for overloading constructors used in on-the-fly instantiations.

- **struct RecastBaseConstructor**
  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 BiStream**
  The binary input stream class. Overloads the >> operator for all data types.

- **class BoStream**
  The binary output stream class. Overloads the << operator for all data types.

- **class Constraints**
  Base class for the variable constraints class hierarchy.

- **class Graphics**
  The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics as well as tabular cataloging of data for post-processing with Matlab, Tecplot, etc.

- **class Interface**
  Base class for the interface class hierarchy.
• class **Iterator**
  
  *Base class for the iterator class hierarchy.*

• class **LeastSq**
  
  *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 **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 **ResponseRep**
  
  *Container class for response functions and their derivatives. ResponseRep provides the body class.*

• class **Response**
  
  *Container class for response functions and their derivatives. Response provides the handle class.*

• class **Strategy**
  
  *Base class for the strategy class hierarchy.*

• class **Variables**
  
  *Base class for the variables class hierarchy.*

• class **Verification**
  
  *Base class for managing common aspects of verification studies.*

• class **DataFitSurrModel**
  
  *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 DataStrategyRep
  Body class for strategy specification data.

- class DataStrategy
  Handle class for strategy specification data.

- 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 DOTOptimizer
  Wrapper class for the DOT optimization library.

- class EffGlobalMinimizer
  Implementation of Efficient Global Optimization/Least Squares algorithms.

- class EfficientSubspaceMethod
  Efficient Subspace Method (ESM), as proposed by Hany S. Abdel-Khalik.

- class EmbeddedHybridStrategy
Strategy for closely-coupled hybrid minimization, typically involving the embedding of local search methods within
global search methods.

- class ExperimentData
- class ForkApplicInterface
  Derived application interface class which spawns simulation codes using fork/execvp/waitpid.
- class FSUDesignCompExp
  Wrapper class for the FSU/ACE 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 HierarchSurrModel
  Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying
  fidelity).
- class HybridStrategy
  Base class for hybrid minimization strategies.
- class JEGAOptimizer
  A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).
- class MatlabInterface
- 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 MPIPackBuffer
  Class for packing MPI message buffers.
- class MPIUnpackBuffer
  Class for unpacking MPI message buffers.
- class NCSUOptimizer
  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.
• class **NIDRProblemDescDB**
  
  *The derived input file database utilizing the new IDR parser.*

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

• class **NL2SOLLeastSq**
  
  *Wrapper class for the NL2SOL nonlinear least squares library.*

• class **NLPQLPOptimizer**
  
  *Wrapper class for the NLPQLP optimization library, Version 2.0.*

• class **NLSSOLLeastSq**
  
  *Wrapper class for the NLSSOL nonlinear least squares library.*

• class **NomadOptimizer**
  
  *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 NonDGPImpSampling
  Class for the Gaussian Process-based Importance Sampling method.

• class NonDGPMSABayesCalibration
  Generates posterior distribution on model parameters given experiment data.

• class NonDIncremLHSSampling
  Performs incremental LHS sampling for uncertainty quantification.

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

• class NonDLHSSampling
  Performs LHS and Monte Carlo sampling for uncertainty quantification.

• class NonDLHSSingleInterval
  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 NonDLocalSingleInterval
Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic 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 NonDQUESOBayesCalibration**

  Bayesian inference using the QUESO library from UT Austin.

- **class NonDReliability**

  Base class for the reliability methods within DAKOTA/UQ.

- **class NonDSampling**

  Base class for common code between NonDLHSSampling, NonDIncremLHSSampling, and NonDAdaptImpSampling.

- **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 NonlinearCGOptimizer**

- **class NPSOLOptimizer**

  Wrapper class for the NPSOL optimization library.

- **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 ParallelLibrary**

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

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

• 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.*

• 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 **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 **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 run_iterator 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** ScilabInterface

- **class** SensAnalysisGlobal
  
  *Class for a utility class containing correlation calculations and variance-based decomposition.*

- **class** SequentialHybridStrategy
  
  *Strategy for sequential hybrid minimization using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.*

- **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** SingleMethodStrategy
  
  *Simple fall-through strategy for running a single iterator on a single model.*

- **class** SingleModel
  
  *Derived model class which utilizes a single interface to map variables into responses.*

- **class** SNLLBase
  
  *Base class for OPT++ optimization and least squares methods.*

- **class** SNLLLeastSq
  
  *Wrapper class for the OPT++ optimization library.*

- **class** SNLLOptimizer
  
  *Wrapper class for the OPT++ optimization library.*

- **class** SOLBase
  
  *Base class for Stanford SOL software.*

- **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.*
12.1 Dakota Namespace Reference

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

- **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 HierarchSurrModel).

- **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.

**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 > **IntVector**
- typedef Teuchos::SerialDenseMatrix< int, int > **IntMatrix**
- typedef std::deque< bool > **BoolDeque**
- typedef boost::dynamic_bitset< unsigned long > **BitArray**
- typedef std::vector< BoolDeque > **BoolDequeArray**
- typedef std::vector< Real > **RealArray**
- typedef std::vector< RealArray > **Real2DArray**
- typedef std::vector< int > **IntArray**
- typedef std::vector< IntArray > **Int2DArray**
- typedef std::vector< short > **ShortArray**
- typedef std::vector< unsigned short > **UShortArray**
• typedef std::vector<UShortArray> UShort2DArray
• typedef std::vector<UShort2DArray> UShort3DArray
• typedef std::vector<SizetArray> Sizet2DArray
• typedef boost::multi_array_types::index_range idx_range
• typedef boost::multi_array<String, 1> StringMultiArray
• typedef StringMultiArray::array_view<1>::type StringMultiArrayView
• typedef StringMultiArray::const_array_view<1>::type StringMultiArrayConstView
• typedef StringMultiArray::array_view<1>::type StringMultiArrayView
• typedef StringMultiArray::const_array_view<1>::type StringMultiArrayConstView
• typedef std::vector<RealVector> RealVectorArray
• typedef std::vector<RealMatrix> RealMatrixArray
• typedef std::vector<RealSymMatrix> RealSymMatrixArray
• 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::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<Model> ModelList
• typedef std::list<Iterator> IteratorList
• typedef std::pair<int, int> IntIntPair
• 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::vector<RealSet>  RealSetArray
• typedef std::vector<IntSet>  IntSetArray
• typedef std::vector<UShortSet>  UShortSetArray
• typedef std::map<int, int>  IntIntMap
• typedef std::map<int, short>  IntShortMap
• typedef std::map<int, Real>  RealRealMap
• typedef std::vector<RealRealMap>  RealRealMapArray
• typedef std::map<int, RealVector>  IntRealVectorMap
• typedef std::map<int, Variables>  IntVariablesMap
• typedef std::map<int, Response>  IntResponseMap
• typedef std::map<int, size_t>  IntArraySizetMap
• typedef std::multimap<RealRealPair, ParamResponsePair>  RealPairPRPMultiMap
• 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 ModelList::iterator ModelLIter
• typedef ModelList::reverse_iterator ModelLRevIter
• typedef IteratorList::iterator IterLIter
• typedef std::list<ParallelLevel>::iterator ParLevLIter
• typedef std::list<ParallelConfiguration>::iterator ParConfigLIter
• typedef IntSet::iterator ISIter
• typedef IntSet::const_iterator ISCIter
• 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 RealRealMap::iterator RRMIter
• typedef RealRealMap::const_iterator RRMCIter
• typedef RealRealMap::const_iterator RRMCIter
• typedef IntRealVectorMap::iterator IntRDVMIter
typedef IntRealVectorMap::const_iterator IntRDVMCIter
typedef IntActiveSetMap::iterator IntASMIter
typedef IntVariablesMap::iterator IntVarsMIter
typedef IntVariablesMap::const_iterator IntVarsMCIter
typedef IntResponseMap::iterator IntRespMIter
typedef IntResponseMap::const_iterator IntRespMCIter
typedef int(* ftw_fn)(const char *file, const struct stat *, int ftype, int depth, void *v)
typedef struct dirent dirent
typedef struct Dakota::Cbuf Cbuf
typedef struct Dakota::Buf Buf
typedef struct Dakota::Finfo Finfo
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> StrStrSizet
Iterator unique ID: <method_name, method_id, exec_num>.

typedef void(*)(dl_find_optimum_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 unsigned char u_char

typedef unsigned short u_short
typedef unsigned int u_int
typedef unsigned long u_long
typedef long long long_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 int MPI_Comm

typedef void MPI_Request

typedef bmi::multi_index_container< Dakota::ParamResponsePair, bmi::indexed_by< bmi::ordered_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_iterator< ordered >::type PRPCacheOCIter

typedef PRPCache::index_iterator< hashed >::type PRPCacheHIter

typedef PRPCache::index_iterator< hashed >::type PRPCacheHCIter

typedef PRPCacheOIter PRPCacheIter

typedef PRPCacheOCIter PRPCacheCIter

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_iterator< ordered >::type PRPQueueOCIter

typedef PRPQueue::index_iterator< hashed >::type PRPQueueHIter

typedef PRPQueue::index_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).

Enumerations

enum {
    COBYLA, DIRECT, EA, MS,
    PS, SW, BETA
}
enum {
    sFTW_F, sFTW_SL, sFTW_D, sFTW_DP,
    sFTW_DNR, sFTW_O, sFTW_NS }
enum {
    sFTWret_OK, sFTWret_quit, sFTWret_skipdir, sFTWret_Follow,
    sFTWret_mallocfailure }
enum { OBJECTIVE, INEQUALITY_CONSTRAINT, EQUALITY_CONSTRAINT }

define algebraic function types

enum {
    SILENT_OUTPUT, QUIET_OUTPUT, NORMAL_OUTPUT, VERBOSE_OUTPUT,
    DEBUG_OUTPUT }
enum { STD_NORMAL_U, STD_UNIFORM_U, ASKEY_U, EXTENDED_U }
enum { DEFAULT_INTERPOLANT, NODAL_INTERPOLANT, HIERARCHICAL_INTERPOLANT }
enum { DEFAULT_COVARIANCE, NO_COVARIANCE, DIAGONAL_COVARIANCE, FULL_COVARIANCE }
enum { NO_INT_REFINE, IS, AIS, MMAIS }
enum { PROBABILITIES, RELIABILITIES, GEN_RELIABILITIES }
enum { COMPONENT = 0, SYSTEM_SERIES, SYSTEM_PARALLEL }
enum { CUMULATIVE, COMPLEMENTARY }
enum { DEFAULT_LS = 0, SVD_LS, EQ_CON_LS }
enum {
    NO_EMULATOR, POLYNOMIAL_CHAOS, STOCHASTIC_COLLOCATION, GAUSSIAN_PROCESS,
    KRYSTOL } 
enum { IGNORE_RANKS, SET_RANKS, GET_RANKS, SET_GET_RANKS }
enum {
    UNCERTAIN, UNCERTAIN_UNIFORM, ALEATORY_UNCERTAIN, ALEATORY_UNIFORM,
    EPISTEMIC_UNCERTAIN, EPISTEMIC_UNCERTAIN_UNIFORM, ACTIVE, ACTIVE_UNIFORM,
    ALL, ALL_UNIFORM }
enum {
    MV, AMV_X, AMV_U, AMV_PLUS_X,
    AMV_PLUS_U, TANA_X, TANA_U, NO_APPROX }
enum { BREITUNG, HORENACK, HONG }
enum { EGRA_X, EGRA_U }
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, 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 }  
  
  define special values for SurrogateModel::responseMode  

• enum {  
  NO_CORRECTION = 0, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, COMBINED_CORRECTION }  
  
  define special values for approxCorrectionType  

• enum {  
  DEFAULT_DOMAIN, RELAXED_DOMAIN, MIXED_DOMAIN }  
• enum {  
  DEFAULT_VIEW, ALL_VIEW, DESIGN_VIEW, UNCERTAIN_VIEW,  
  ALEATORY_UNCERTAIN_VIEW, EPISTEMIC_UNCERTAIN_VIEW, STATE_VIEW }  
• enum {  
  EMPTY, 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 {  
  CONTINUOUS_DESIGN, DISCRETE_DESIGN_RANGE, DISCRETE_DESIGN_SET_INT,  
  DISCRETE_STATE_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, CONTINUOUS_INTERVAL_UNCERTAIN, DISCRETE_INTERVAL_UNCERTAIN,  
  DISCRETE_UNCERTAIN_SET_INT, DISCRETE_UNCERTAIN_SET_REAL, CONTINUOUS_STATE, DISCRETE_STATE_RANGE,  
  DISCRETE_STATE_SET_INT, DISCRETE_STATE_SET_REAL }
• 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, MF_ROSENBROCK,
    ROSENBROCK, GERSTNER, SCALABLEGERSTNER, LOGNORMAL_RATIO,
    MULTIMODAL, PLUGINROSENBROCK, PLUGINTEXT_BOOK, SHORT_COLUMN,
    LF_SHORT_COLUMN, MF_SHORT_COLUMN, SIDE_IMPACT_COST, SIDE_IMPACT_PERFORMANCE,
    SOBOL_RATIONAL, SOBOL_G_FUNCTION, SOBOLISHIGAMI, STEEL_COLUMN_COST,
    STEEL_COLUMN_PERFORMANCE, TEXT_BOOK, TEXT_BOOK1, TEXT_BOOK2,
    TEXT_BOOK3, TEXT_BOOK_OUU, SCALABLETEXT_BOOK, SCALABLEMONOMIALS,
    HERBIE, SMOOTHHERBIE, SHUBERT, SALINAS,
    MODELCENTER }
    
    enumeration of possible direct driver types (to index to names)

• enum local_data_t { VARIABLES_MAP = 1, VARIABLES_VECTOR = 2 }
    
    enumeration for how local variables are stored (values must employ a bit representation)

• enum sigtype { NO_SIGMA, SCALAR_SIGMA, COVARIANCE_MATRIX }

• enum edtype { SCALAR_DATA, FUNCTIONAL_DATA }

• 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_Nkinds = 5 }

• enum { DAURVar_histogram_point = 0, DAURVar_Nkinds = 1 }

• enum { CEUVar_interval = 0, CEUVar_Nkinds = 1 }
• enum { DEUIVar_interval = 0, DEUIVar_set_int = 1, DEUIVar_Nkinds = 2 }
• enum { DEURVar_set_real = 0, DEURVar_Nkinds = 1 }
• enum {
  DiscSetVar_design_set_int = 0, DiscSetVar_design_set_real = 1, DiscSetVar_state_set_int = 2,
  DiscSetVar_state_set_real = 3,
  DiscSetVar_Nkinds = 4 }
• enum { N_VLR = 4 }
• enum { N_VLI = 2 }
• enum { FULL_TENSOR, FILTERED_TENSOR, RANDOM_TENSOR }
• enum CG_UPDATETYPE {
  CG_STEEPEST, CG_FLETCHER_REEVES, CG_POLAK_RIBIERE, CG_POLAK_RIBIERE_-PLUS,
  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 {
  LIST = 1, VECTOR_SV, VECTOR_FP, CENTERED,
  MULTIDIM }
• enum { ESTIMATE_ORDER = 1, CONVERGE_ORDER, CONVERGE_QOI }
• enum EvalType { NLEEvaluator, CONEvaluator }
  enumeration for the type of evaluator function
• enum {
  TH_SILENT_OUTPUT, TH_QUIET_OUTPUT, TH_NORMAL_OUTPUT, TH_VERBOSE_-OUTPUT,
  TH_DEBUG_OUTPUT }

Functions

• CommandShell & flush (CommandShell &shell)
  convenient shell manipulator function to “flush” the shell
• template<typename T>
  std::ostream & operator<< (std::ostream &, const std::set<T> &data)
    global std::ostream insertion operator for std::set
• template<typename T>
  MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, std::set<T> &data)
    global MPIUnpackBuffer extraction operator for std::set
• template<typename T>
  MPIPackBuffer & operator<< (MPIPackBuffer &s, const std::set<T> &data)
  
  global MPIPackBuffer insertion operator for std::set

• template<typename KeyT, typename ValueT>
  std::ostream & operator<<(std::ostream &s, const std::map<KeyT, ValueT> &data)
  
  global std::ostream insertion operator for std::map

• template<typename OrdinalType, typename ScalarType>
  MPIPackBuffer & operator<< (MPIPackBuffer &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &data)
  
  global MPIPackBuffer insertion operator for Teuchos::SerialDenseVector

• template<typename OrdinalType, typename ScalarType>
  MPIPackBuffer & operator<<(MPIPackBuffer &s, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &data)
  
  global MPIPackBuffer insertion operator for Teuchos::SerialDenseMatrix

• template<typename OrdinalType, typename ScalarType>
  MPIPackBuffer & operator<<(MPIPackBuffer &s, const Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &data)
  
  global MPIPackBuffer insertion operator for Teuchos::SerialSymDenseMatrix

• template<typename OrdinalType, typename ScalarType>
  MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &data)
  
  global MPIUnpackBuffer extraction operator for Teuchos::SerialDenseVector

• template<typename OrdinalType, typename ScalarType>
  MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &data)
  
  global MPIUnpackBuffer extraction operator for Teuchos::SerialDenseMatrix

• template<typename OrdinalType, typename ScalarType>
  MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &data)
  
  global MPIUnpackBuffer extraction operator for Teuchos::SerialSymDenseMatrix

• template<typename OrdinalType, typename ScalarType>
  void read_data (std::istream &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v)
standard istream extraction operator for full SerialDenseVector

- template<typename OrdinalType , typename ScalarType >
  void read_data (std::istream &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArray &label_array)

  standard istream extraction operator for full SerialDenseVector with labels

- template<typename OrdinalType , typename ScalarType >
  void read_data (std::istream &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArrayView label_array)

  standard istream extraction operator for full SerialDenseVector with labels

- template<typename OrdinalType , typename ScalarType >
  void read_data_partial (std::istream &s, size_t start_index, size_t num_items, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v)

  standard istream extraction operator for partial SerialDenseVector

- template<typename OrdinalType , typename ScalarType >
  void read_data_partial (std::istream &s, size_t start_index, size_t num_items, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArray &label_array)

  standard istream extraction operator for partial SerialDenseVector with labels

- template<typename OrdinalType , typename ScalarType >
  void read_data_partial (std::istream &s, size_t start_index, size_t num_items, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArrayView label_array)

  standard istream extraction operator for partial SerialDenseVector with labels

- template<typename OrdinalType , typename ScalarType >
  void read_data_tabular (std::istream &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v)

  tabular istream extraction operator for full SerialDenseVector

- template<typename OrdinalType , typename ScalarType >
  void read_data_partial_tabular (std::istream &s, size_t start_index, size_t num_items, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v)

  tabular istream extraction operator for partial SerialDenseVector

- template<typename OrdinalType , typename ScalarType >
  void read_data_annotated (std::istream &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArray &label_array)

  annotated istream extraction operator for full SerialDenseVector with labels

- template<typename OrdinalType , typename ScalarType >
  void read_data_annotated (std::istream &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArrayView label_array)

  annotated istream extraction operator for full SerialDenseVector with labels
Namespace Documentation

- template<typename OrdinalType, typename ScalarType >
  void write_data (std::ostream &s, const ScalarType *, OrdinalType len)
  
  standard ostream insertion operator for full SerialDenseVector

- template<typename OrdinalType, typename ScalarType >
  void write_data (std::ostream &s, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v)
  
  standard ostream insertion operator for full SerialDenseVector

- template<typename OrdinalType, typename ScalarType >
  void write_data (std::ostream &s, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, const StringMultiArray &label_array)
  
  standard ostream insertion operator for full SerialDenseVector with labels

- template<typename OrdinalType, typename ScalarType >
  void write_data (std::ostream &s, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArrayConstView label_array)
  
  standard ostream insertion operator for full SerialDenseVector with labels

- template<typename OrdinalType, typename ScalarType >
  void write_data (std::ostream &s, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, const StringArray &label_array)
  
  standard ostream insertion operator for full SerialDenseVector with alternate labels

- template<typename OrdinalType, typename ScalarType >
  void write_data_aprepro (std::ostream &s, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, const StringMultiArray &label_array)
  
  aprepro ostream insertion operator for full SerialDenseVector with labels

- template<typename ScalarType >
  void write_data_partial (std::ostream &s, size_t start_index, size_t num_items, const std::vector<ScalarType> &v)
  
  standard ostream insertion operator for partial SerialDenseVector

- template<typename OrdinalType, typename ScalarType >
  void write_data_partial (std::ostream &s, size_t start_index, size_t num_items, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v)
  
  standard ostream insertion operator for partial SerialDenseVector

- template<typename OrdinalType, typename ScalarType >
  void write_data_partial (std::ostream &s, size_t start_index, size_t num_items, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, const StringMultiArray &label_array)
  
  standard ostream insertion operator for partial SerialDenseVector with labels

- template<typename OrdinalType, typename ScalarType >
  void write_data_partial_aprepro (std::ostream &s, size_t start_index, size_t num_items, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, const StringMultiArray &label_array)

Dakota Version 5.4 Developers Manual generated on November 7, 2013
aprepro ostream insertion operator for partial SerialDenseVector with labels

- template<typename OrdinalType, typename ScalarType>
  void write_data_annotated (std::ostream &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, const StringMultiArray &label_array)
  annotated ostream insertion operator for full SerialDenseVector with labels

- template<typename OrdinalType, typename ScalarType>
  void write_data_tabular (std::ostream &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v)
  tabular ostream insertion operator for full SerialDenseVector

- template<typename OrdinalType, typename ScalarType>
  void write_data_tabular (std::ostream &s, const ScalarType *ptr, OrdinalType num_items)
  tabular ostream insertion operator for full SerialDenseVector

- template<typename OrdinalType, typename ScalarType>
  void write_data_partial_tabular (std::ostream &s, size_t start_index, size_t num_items, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v)
  tabular ostream insertion operator for partial SerialDenseVector

- void write_data_tabular (std::ostream &s, const StringArray &sa)
  tabular ostream insertion operator for vector of strings

- void write_data_tabular (std::ostream &s, StringMultiArrayConstView ma)
  tabular ostream insertion operator for view of StringMultiArray

- template<typename OrdinalType, typename ScalarType>
  std::istream & operator>> (std::istream &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &data)
  global std::istream extraction operator for SerialDenseVector

- template<typename OrdinalType, typename ScalarType>
  std::ostream & operator<< (std::ostream &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &data)
  global std::ostream insertion operator for SerialDenseVector

- template<typename OrdinalType, typename ScalarType>
  void read_data (BiStream &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, StringMultiArray &label_array)
  standard binary stream extraction operator for full SerialDenseVector with labels

- template<typename OrdinalType, typename ScalarType>
  void read_data (BiStream &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, StringMultiArrayView label_array)
  standard binary stream extraction operator for full SerialDenseVector with labels
- `template<typename OrdinalType, typename ScalarType>
  void write_data (BoStream &s, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, const StringMultiArray &label_array)
  
  *default* binary stream insertion operator for full SerialDenseVector with labels

- `template<typename OrdinalType, typename ScalarType>
  void read_data (MPIUnpackBuffer &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArray &label_array)
  
  *default* MPI buffer extraction operator for full SerialDenseVector with labels

- `template<typename OrdinalType, typename ScalarType>
  void read_data (MPIUnpackBuffer &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArrayView label_array)
  
  *default* MPI buffer extraction operator for full SerialDenseVector with labels

- `template<typename OrdinalType, typename ScalarType>
  void write_data (MPIPackBuffer &s, const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, const StringMultiArray &label_array)
  
  *default* MPI buffer insertion operator for full SerialDenseVector with labels

- `template<typename OrdinalType, typename ScalarType>
  void read_data (std::istream &s, std::vector<Teuchos::SerialDenseVector< OrdinalType, ScalarType > > &va)
  
  *default* istream extraction operator for std::vector of SerialDenseVectors

- `template<typename OrdinalType, typename ScalarType>
  void read_data (std::istream &s, Teuchos::SerialSymDenseMatrix< OrdinalType, ScalarType > &m)
  
  *default* istream extraction operator for SerialSymDenseMatrix

- `template<typename OrdinalType, typename ScalarType>
  void write_data (std::ostream &s, const Teuchos::SerialSymDenseMatrix< OrdinalType, ScalarType > &m, bool brackets, bool row_rtn, bool final_rtn)
  
  formatted ostream insertion operator for SerialSymDenseMatrix

- `template<typename OrdinalType, typename ScalarType>
  void write_data (std::ostream &s, const Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &m, bool brackets, bool row_rtn, bool final_rtn)
  
  formatted ostream insertion operator for SerialDenseMatrix

- `template<typename OrdinalType, typename ScalarType>
  void write_data (std::ostream &s, const Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &m, const StringArray &row_labels, const StringArray &col_labels)
  
  ostream insertion operator for SerialDenseMatrix with row/col labels

- `template<typename OrdinalType, typename ScalarType>
  void read_lower_triangle (std::istream &s, Teuchos::SerialSymDenseMatrix< OrdinalType, ScalarType > &sm)
istream partial specialization for reading the lower triangle of a SerialSymDenseMatrix

- template<typename IStreamType, typename OrdinalType, typename ScalarType>
  void read_lower_triangle (IStreamType &s, Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &sm)

  generic input stream template for reading the lower triangle of a SerialSymDenseMatrix

- template<typename OrdinalType, typename ScalarType>
  void write_lower_triangle (std::ostream &s, const Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &sm, bool row_rtn)

  ostream version for writing the lower triangle of a SerialSymDenseMatrix

- template<typename OStreamType, typename OrdinalType, typename ScalarType>
  void write_lower_triangle (OStreamType &s, const Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &sm)

  generic output stream template for writing the lower triangle of a SerialSymDenseMatrix

- template<typename OrdinalType, typename ScalarType>
  void read_col_vector_trans (std::istream &s, OrdinalType col, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)

  istream partial specialization for reading a column vector of a SerialDenseMatrix

- template<typename IStreamType, typename OrdinalType, typename ScalarType>
  void read_col_vector_trans (IStreamType &s, OrdinalType col, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)

  generic input stream template for reading a column vector of a SerialDenseMatrix

- template<typename OrdinalType, typename ScalarType>
  void write_col_vector_trans (std::ostream &s, OrdinalType col, OrdinalType num_items, bool brackets, bool break_line, bool final_rtn, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)

  ostream insertion operator for a column vector from a SerialDenseMatrix

- template<typename OStreamType, typename OrdinalType, typename ScalarType>
  void write_col_vector_trans (OStreamType &s, OrdinalType col, OrdinalType num_items, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)

  ostream insertion operator for a column vector from a SerialDenseMatrix

- template<typename OStreamType, typename OrdinalType, typename ScalarType>
  void write_col_vector_trans (OStreamType &s, OrdinalType col, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)

  ostream insertion operator for a column vector from a SerialDenseMatrix
Namespace Documentation

- template<typename ArrayT>
  void array_read (std::istream &s, ArrayT &v)
  
  *read array from std::istream*

- template<typename ArrayT>
  void array_write (std::ostream &s, const ArrayT &v)
  
  *write array to std::ostream*

- template<typename ListT>
  void list_write (const ListT &l, std::ostream &s)
  
  *write list to std::ostream*

- template<typename T>
  std::istream & operator>>(std::istream &s, std::vector<T> &data)
  
  *global std::istream extraction operator for std::vector*

- template<typename T>
  std::ostream & operator<<(std::ostream &s, const std::vector<T> &data)
  
  *global std::ostream insertion operator for std::vector*

- template<typename T>
  std::ostream & operator<<(std::ostream &s, const std::list<T> &data)
  
  *global std::ostream insertion operator for std::list*

- template<typename ArrayT>
  void array_write (std::ostream &s, const ArrayT &v, const std::vector<String> &label_array)
  
  *write array to std::ostream with labels*

- template<typename ArrayT>
  void array_write_aprepro (std::ostream &s, const ArrayT &v, const std::vector<String> &label_array)
  
  *write array to std::ostream (APREPRO format)*

- template<typename ArrayT>
  void array_write_annotated (std::ostream &s, const ArrayT &v, bool write_len)
  
  *Write array to ostream as a row vector; precede with length if write_len = true.*

- bool operator==(const ShortArray &dsa1, const ShortArray &dsa2)
  
  *equality operator for ShortArray*

- bool operator==(const StringArray &dsa1, const StringArray &dsa2)
  
  *equality operator for StringArray*

- bool operator==(const SizetArray &sa, SizetMultiArrayConstView smav)
  
  *equality operator for SizetArray and SizetMultiArrayConstView*

- Real rel_change_L2 (const RealVector &curr_rv, const RealVector &prev_rv)
  
  *Computes relative change between RealVectors using Euclidean L2 norm.*
• Real \texttt{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.

• bool \texttt{operator==} (const IntArray \&dia1, const IntArray \&dia2)  
  equality operator for IntArray

• bool \texttt{operator!=} (const IntArray \&dia1, const IntArray \&dia2)  
  inequality operator for IntArray

• bool \texttt{operator!=} (const ShortArray \&dsa1, const ShortArray \&dsa2)  
  inequality operator for ShortArray

• bool \texttt{operator!=} (const StringArray \&dsa1, const StringArray \&dsa2)  
  inequality operator for StringArray

• bool \texttt{operator!=} (const SizetArray \&sa, SizetMultiArrayConstView smav)  
  inequality operator for StringArray

• std::string \texttt{strtolower} (const std::string \&s)  
  Return lowercase copy of string s.

• bool \texttt{strbegins} (const std::string \&input, const std::string \&test)  
  Return true if input string begins with string test.

• bool \texttt{strends} (const std::string \&input, const std::string \&test)  
  Return true if input string ends with string test.

• bool \texttt{strcontains} (const std::string \&input, const std::string \&test)  
  Return true if input string contains string test.

• void \texttt{build\_label} (String \&label, const String \&root\_label, size\_t tag)  
  create a label by appending a numerical tag to the root\_label

• void \texttt{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 \texttt{build\_label}().

• void \texttt{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 \texttt{build\_label}().

• void \texttt{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 \texttt{build\_label}().

• void \texttt{copy\_row\_vector} (const RealMatrix \&m, RealMatrix::ordinalType i, std::vector< Real > \&row)
Copies a row of a Teuchos_SerialDenseMatrix<int, Real> to std::vector<Real>.

- **template<typename T>**
  ```
  void copy_data (const std::vector<T> &vec, T *ptr, const size_t ptr_len)
  ```
  
- **template<typename T>**
  ```
  void copy_data (const T *ptr, const size_t ptr_len, std::vector<T> &vec)
  ```

- **template<typename OrdinalType, typename ScalarType>**
  ```
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdva, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm, OrdinalType nr, OrdinalType nc)
  ```

- **template<typename T>**
  ```
  void copy_data (std::map<int, T> &im, std::vector<T> &da)
  ```

- **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 std::vector<ScalarType>

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const std::vector<ScalarType> &da, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv)

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv)

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const ScalarType *ptr, const OrdinalType ptr_len, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv, ScalarType *ptr, const OrdinalType ptr_len)

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv1, OrdinalType start_index1, OrdinalType num_items, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv2)

- template<typename OrdinalType, typename ScalarType>
  void copy_data_partial (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv1, OrdinalType start_index1, OrdinalType num_items, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv2)

- template<typename OrdinalType, typename ScalarType>
  void copy_data_partial (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv1, OrdinalType start_index1, OrdinalType num_items, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv2, OrdinalType start_index2)

- template<typename OrdinalType, typename ScalarType, size_t>
  void copy_data_partial (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv1, OrdinalType start_index1, OrdinalType num_items, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv2, OrdinalType start_index2,
  std::vector<ScalarType> &da2, size_t start_index2)
• template<typename T>
  void \textit{copy\_data\_partial}\ (\textit{const std::vector<T> \&da1, std::vector<T> \&da2, size_t start\_index2})
  \textit{copy all of first Array\langle T\rangle to portion of second Array\langle T\rangle}

• template<typename T>
  void \textit{copy\_data\_partial}\ (\textit{const std::vector<T> \&da, boost::multi\_array<T, 1> \&bma, size_t start\_index\_bma})
  \textit{copy all of first Array\langle T\rangle to portion of boost::multi\_array<T, 1>}

• template<typename T>
  void \textit{copy\_data\_partial}\ (\textit{const std::vector<T> \&da, size_t start\_index1, size_t num\_items, std::vector<T> \&da2, size_t start\_index2})
  \textit{copy portion of first Array\langle T\rangle to portion of second Array\langle T\rangle}

• void \textit{merge\_data\_partial}\ (\textit{const Int\_vector \&d\_vec, Real\_vector \&m\_vec, size_t start\_index\_ma})
  \textit{merge a discrete integer vector into a single continuous vector}

• void \textit{merge\_data\_partial}\ (\textit{const Int\_vector \&d\_vec, Real\_array \&m\_array, size_t start\_index\_ma})
  \textit{merge a discrete integer vector into a single continuous array}

• template<typename Ordinal\_type , typename Scalar\_type>
  const Scalar\_type & \textit{set\_index\_to\_value}\ (Ordinal\_type index, \textit{const std::set<Scalar\_type> \&values})
  \textit{retrieve the set value corresponding to the passed index}

• template<typename Scalar\_type>
  size_t \textit{set\_value\_to\_index}\ (const Scalar\_type \&value, \textit{const std::set<Scalar\_type> \&values})
  \textit{calculate the set index corresponding to the passed value}

• template<typename Ordinal\_type , typename Key\_type , typename Value\_type>
  const Key\_type & \textit{map\_index\_to\_key}\ (Ordinal\_type index, \textit{const std::map<Key\_type, Value\_type> \&pairs})
  \textit{retrieve the set value corresponding to the passed index}

• template<typename Ordinal\_type , typename Key\_type , typename Value\_type>
  const Value\_type & \textit{map\_index\_to\_value}\ (Ordinal\_type index, \textit{const std::map<Key\_type, Value\_type> \&pairs})
  \textit{retrieve the set value corresponding to the passed index}

• template<typename Key\_type , typename Value\_type>
  void \textit{map\_keys\_to\_set}\ (\textit{const std::map<Key\_type, Value\_type> \&source\_map, std::set<Key\_type> \&target\_set})
  \textit{calculate the map index corresponding to the passed key}

• template<typename Key\_type , typename Value\_type>
  size_t \textit{map\_key\_to\_index}\ (const Key\_type \&key, \textit{const std::map<Key\_type, Value\_type> \&pairs})
calculate the map index corresponding to the passed key

- 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>
  ContainerType::size_type find_index (const ContainerType &c, const typename ContainerType::value_type &search_data)
  generic find_index (inactive)

- template<typename MultiArrayType, typename DakArrayType>
  void copy_data (const MultiArrayType &ma, DakArrayType &da)
  generic copy (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>
  ListT::size_type find_index (const ListT &l, const typename ListT::value_type &val)
  compute the index of an entry within a std::list

- void copy_data (SizetMultiArrayConstView ma, SizetArray &da)
  copy boost::multi_array view to Array

- void copy_data (StringMultiArrayConstView ma, StringArray &da)
  copy boost::multi_array view to Array

- 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 DakContainerType>
  bool contains (const DakContainerType &v, const typename DakContainerType::value_type &val)
  return true if the item val appears in container v

- void dak_sigcatch (int sig)
- void `start_dakota_heartbeat` (int seconds)
- int `sftw` (const char *name, int(*)(fn)(const char *file, const struct stat *, int ftype, int depth, void *v), void *v)
- static int `ftw1` (char *name, size_t namelen, size_t namemaxlen, ftw_fn fn, int, void *v)
- static int `compar` (const void *a, const void *b)
- static int `dodir` (DIR *dir, char *name, size_t namelen, size_t namemaxlen, ftw_fn fn, int depth, void *v, struct stat *sb)
- int `sftw` (const char *name, ftw_fn fn, void *v)
- static int `Symlink` (const char *from, const char *to)
- static int `my_recrm` (const char *file, const struct stat *sb, int ftype, int depth, void *v)
- int `rec_rmdir` (const char *name)
- static void `buf_incr` (Buf *b, size_t Lt)
- int `my_cp` (const char *file, const struct stat *sb, int ftype, int depth, void *v)
- int `rec_cp` (const char *from, const char *todir, int copy, int flatten, int replace)
- static char * `pathsimp` (char *t0)
- void `get_npath` (int appdrive, std::string *pnpath)
- void `workdir_adjust` (const std::string &workdir)
- std::string `get_cwd`()
  
  *Portability adapter for getcwd.

- void `putenv_impl` (const char *name_and_value)
  
  *Utility function from boost/test, not available in the DAKOTA snapshot.

- void `abort_handler` (int code)
  
  *Global function which handles serial or parallel aborts

- template<typename T>
  T `abort_handler_t` (int code)
- 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 &, const std::string &)
  
  *Create MetaDataValueType from the passed strings
- **MetaDataValueType**: `make_metadatavalue` (const std::string & set1, const std::string & set2)
  - create MetaDataValueType from the passed strings

- **MetaDataValueType**: `make_metadatavalue` (const std::string & set1, const std::string & set2)
  - create MetaDataValueType from the passed strings

- **bool**: `operator==` (const ActiveSet & set1, const ActiveSet & set2)
  - equality operator for ActiveSet

- **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&).

- **BiStream &**: `operator>>` (BiStream & s, ActiveSet & set)
  - BiStream extraction operator for ActiveSet. Calls read(BiStream&).

- **BoStream &**: `operator<<` (BoStream & s, const ActiveSet & set)
  - BoStream insertion operator for ActiveSet. Calls write(BoStream&).

- **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

- **template<class ArrayT>**
  - void `array_read` (BiStream & s, ArrayT & v)
    - Read an array from BiStream, s.

- **template<class ArrayT>**
  - void `array_write` (BoStream & s, const ArrayT & v)
    - Write an array to BoStream, s.

- **template<class ArrayT>**
  - BiStream & `operator>>` (BiStream & s, ArrayT & data)
    - global BiStream extraction operator for generic "array" container

- **template<class ArrayT>**
  - BoStream & `operator<<` (BoStream & s, const ArrayT & data)
Namespace Documentation

global `BoStream` insertion operator for generic "array" container

- `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`

- `bool interface_id_compare (const Interface &interface, 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 ResponseRep &rep1, const ResponseRep &rep2)`
  `equality operator for ResponseRep`

- `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&).`

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

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

- `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)`
  `equality operator for Response`

- `bool operator!=(const Response &resp1, const Response &resp2)`
  `inequality operator for Response`
• std::string `re_match` (const std::string &token, const boost::regex &re)
  
  Global utility function to ease migration from CielRegExp to Boost.Regexp.

• bool `operator==` (const Variables &vars1, const Variables &vars2)
  
  equality operator for Variables

• bool `binary_equal_to` (const Variables &vars1, const Variables &vars2)
  
  binary_equal_to (since 'operator==' is not suitable for boost[hashed lookup]

• std::size_t `hash_value` (const Variables &vars)
  
  hash_value for Variables - required by the new BMI hash_set of PRPairs

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

• std::istream & `operator>>(std::istream &, Variables &vars)
  
  std::istream extraction operator for Variables.

• std::ostream & `operator<<(std::ostream &, const Variables &vars)
  
  std::ostream insertion operator for Variables.

• BiStream & `operator>>(BiStream &, Variables &vars)
  
  BiStream extraction operator for Variables.

• BoStream & `operator<<(BoStream &, const Variables &vars)
  
  BoStream insertion operator for Variables.

• MPIUnpackBuffer & `operator>>(MPIUnpackBuffer &, Variables &vars)
  
  MPIUnpackBuffer extraction operator for Variables.

• MPIPackBuffer & `operator<<(MPIPackBuffer &, 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>

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

  ScalarType1 will be Real, ScalarType2 will be int, and ScalarType3 may be int or Real, but written for arbitrary types.

• template<typename ScalarType>

  void `write_ordered` (std::ostream &, const SizetArray &comp_totals, const std::vector< ScalarType > &c_vector, const std::vector< ScalarType > &di_vector, const std::vector< ScalarType > &dr_vector)
ScalarType1 will be Real, ScalarType2 will be int, and ScalarType3 may be int or Real, but written for arbitrary types.

- 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.

- 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 DataStrategy &data)
  MPIPackBuffer insertion operator for DataStrategy.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataStrategy &data)
  MPIUnpackBuffer extraction operator for DataStrategy.

- std::ostream & operator<<(std::ostream &s, const DataStrategy &data)
std::ostream insertion operator for DataStrategy

- 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 dlsolver_option (Opt_Info *)

- RealVector const * continuous_lower_bounds (Optimizer1 *)

- RealVector const * continuous_upper_bounds (Optimizer1 *)

- RealVector const * nonlinear_ineq_constraint_lower_bounds (Optimizer1 *)

- RealVector const * nonlinear_ineq_constraint_upper_bounds (Optimizer1 *)

- RealVector const * nonlinear_eq_constraint_targets (Optimizer1 *)

- RealVector const * linear_ineq_constraint_lower_bounds (Optimizer1 *)

- RealVector const * linear_ineq_constraint_upper_bounds (Optimizer1 *)

- RealVector const * linear_eq_constraint_targets (Optimizer1 *)

- void ComputeResponses (Optimizer1 *, int mode, int n, double *)

- void GetFuncs (Optimizer1 *, int m0, int m1, double *)

- void GetGrads (Optimizer1 *, int m0, int m1, int n, int is, int js, double *)

- void GetContVars (Optimizer1 *, int n, double *)

- void SetBestContVars (Optimizer1 *, int n, double *)

- void dl_constructor (Optimizer1 *, Dakota_funcs *, dl_find_optimum_t *, dl_destructor_t *)

- static RealVector const * continuous_lower_bounds1 (Optimizer1 *)

- static RealVector const * continuous_upper_bounds1 (Optimizer1 *)

- static RealVector const * nonlinear_ineq_constraint_lower_bounds1 (Optimizer1 *)

- static RealVector const * nonlinear_ineq_constraint_upper_bounds1 (Optimizer1 *)

- static RealVector const * nonlinear_eq_constraint_targets1 (Optimizer1 *)

- static RealVector const * linear_ineq_constraint_lower_bounds1 (Optimizer1 *)

- static RealVector const * linear_ineq_constraint_upper_bounds1 (Optimizer1 *)

- static RealVector const * linear_eq_constraint_targets1 (Optimizer1 *)

- static RealMatrix const * linear_eq_constraint_coeffs1 (Optimizer1 *)

- static RealMatrix const * linear_ineq_constraint_coeffs1 (Optimizer1 *)

- static void ComputeResponses1 (Optimizer1 *, int mode, int n, double *)

- static void GetFuncs1 (Optimizer1 *, int m0, int m1, double *)

- static void GetGrads1 (Optimizer1 *, int m0, int m1, int n, int is, int js, double *)

- static void GetContVars1 (Optimizer1 *, int n, double *)

- static void SetBestContVars1 (Optimizer1 *, int n, double *)

- static void SetBestRespFns1 (Optimizer1 *, int n, double *)
• static double Get_Real1 (Optimizer1 *o, const char *name)
• static int Get_Int1 (Optimizer1 *o, const char *name)
• static bool Get_Bool1 (Optimizer1 *o, const char *name)
• DOTOptimizer * new_DOTOptimizer (Model &model)
• DOTOptimizer * new_DOTOptimizer (NoDBBaseConstructor, Model &model)

void read_historical_data (const std::string &expDataFileName, const std::string &context_message, size_t numExperiments, IntVector &numReplicates, size_t numExpConfigVars, size_t numFunctions, size_t numExpStdDeviationsRead, bool expDataFileAnnotated, bool calc_sigma_from_data, RealMatrix &xObsData, RealMatrixArray &yObsData, RealMatrixArray &yStdData)

Read data in historical format into x, y, sigma matrices.

• Real getdist (const RealVector &x1, const RealVector &x2)
• Real mindist (const RealVector &x, const RealMatrix &xset, int except)
• Real mindistindx (const RealVector &x, const RealMatrix &xset, const IntArray &indx)
• Real getRmax (const RealMatrix &xset)
• int start_grid_computing (char *analysis_driver_script, char *params_file, char *results_file)
• int stop_grid_computing ()
• int perform_analysis (char *iteration_num)
• template<typename T>
   string asstring (const T &val)

Creates a string from the argument val using an ostringstream.

• void run_dakota_data ()

Function to encapsulate the DAKOTA object instantiations for mode 2: direct Data class instantiation.

• PACKBUF (int, MPI_INT) PACKBUF(u_int
• MPI_UNSIGNED PACKBUF (long, MPI_LONG) PACKBUF(u_long
• MPI_UNSIGNEDMPI_UNSIGNED_LONG PACKBUF (short, MPI_SHORT) PACKBUF(u_short
• MPI_UNSIGNED_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_MPI_UNSIGNED_CHAR 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_MPI_UNSIGNED_CHAR PACKSIZE (double, MPI_DOUBLE) PACKSIZE(float

Dakota Version 5.4 Developers Manual generated on November 7, 2013
• MPI_UNSIGNED  MPI_UNSIGNED_LONG  MPI_UNSIGNED_SHORT  MPI_UNSIGNED_CHAR
  MPI_FLOAT int MPIPackSize (const bool &data, const int num=1)
  \textit{return packed size of a bool}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const int &data)
  \textit{insert an int}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const u_int &data)
  \textit{insert a u_int}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const long &data)
  \textit{insert a long}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const u_long &data)
  \textit{insert a u_long}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const short &data)
  \textit{insert a short}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const u_short &data)
  \textit{insert a u_short}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const char &data)
  \textit{insert a char}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const u_char &data)
  \textit{insert a u_char}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const double &data)
  \textit{insert a double}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const float &data)
  \textit{insert a float}

• MPIPackBuffer & operator\texttt{<<} (MPIPackBuffer &buff, const bool &data)
  \textit{insert a bool}

• MPIUnpackBuffer & operator\texttt{>>} (MPIUnpackBuffer &buff, int &data)
  \textit{extract an int}

• MPIUnpackBuffer & operator\texttt{>>} (MPIUnpackBuffer &buff, u_int &data)
  \textit{extract a u_int}

• MPIUnpackBuffer & operator\texttt{>>} (MPIUnpackBuffer &buff, long &data)
  \textit{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<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
12.1 Dakota Namespace Reference

- 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 *)

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

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

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

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

- static void mixed_check2 (size_t 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, Real *rp)

- static void bad_initial_ivalue (const char *kind, int 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_ContinuousState (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, Var_Info *vi)
- static void Vchk_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_WeibullUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_WeibullUnc (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_GumbelUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_GumbelUnc (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_BinomialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_BinomialUnc (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_ExponentialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_ExponentialUnc (DataVariablesRep *dv, size_t offset)
- static bool check_set_keys (size_t num_v, size_t ds_len, const char *kind, IntArray *input_nds, int &avg_minus num_ds)
- static void Vchk_DIset (size_t num_v, const char *kind, IntArray *input_ndsi, IntVector *input_dsi, IntSetArray &dsi_all, IntVector &dsi_init_pt)
- static void Vchk_DSset (size_t num_v, const char *kind, IntArray *input_nds, IntVector *input_dsr, RealVector *input_dsip, RealSetArray &dsr_all, RealVector &dsr_init_pt)
• static void \texttt{Vchk\_DRset} (size\_t num\_v, const char *kind, IntArray *input\_ndsr, RealVector *input\_dsr, RealVector *input\_dsrp, RealRealMapArray &dsr\_vals\_probs)
• static bool \texttt{check\_LUV\_size} (size\_t num\_v, IntVector &L, IntVector &U, IntVector &V, bool aggregate\_LUV, size\_t offset)
• static bool \texttt{check\_LUV\_size} (size\_t num\_v, RealVector &L, RealVector &U, RealVector &V, bool aggregate\_LUV, size\_t offset)
• static void \texttt{Vgen\_Dset} (size\_t num\_v, IntSetArray &sets, IntVector &L, IntVector &U, IntVector &V, bool aggregate\_LUV=false, size\_t offset=0)
• static void \texttt{Vgen\_Dset} (size\_t num\_v, IntRealMapArray &vals\_probs, IntVector &L, IntVector &U, IntVector &V, bool aggregate\_LUV=false, size\_t offset=0)
• static void \texttt{Vgen\_Dset} (size\_t num\_v, RealSetArray &sets, RealVector &L, RealVector &U, RealVector &V, bool aggregate\_LUV=false, size\_t offset=0)
• static void \texttt{Vgen\_Dset} (size\_t num\_v, RealRealMapArray &vals\_probs, RealVector &L, RealVector &U, RealVector &V, RealVector &V, bool aggregate\_LUV=false, size\_t offset=0)
• static void \texttt{Vgen\_Dset} (size\_t num\_v, RealRealMapArray &vals\_probs, RealVector &L, RealVector &U, RealVector &V, RealVector &V, bool aggregate\_LUV=false, size\_t offset=0)
• static void \texttt{Vgen\_Dset} (size\_t num\_v, RealRealMapArray &vals\_probs, RealVector &L, RealVector &U, RealVector &V, RealVector &V, bool aggregate\_LUV=false, size\_t offset=0)
• static void \texttt{Vchk\_DiscreteDesSetInt} (DataVariablesRep *dv, size\_t offset, Var\_Info *vi)
• static void \texttt{Vchk\_DiscreteDesSetReal} (DataVariablesRep *dv, size\_t offset, Var\_Info *vi)
• static void \texttt{Vchk\_DiscreteUncSetInt} (DataVariablesRep *dv, size\_t offset, Var\_Info *vi)
• static void \texttt{Vchk\_DiscreteUncSetReal} (DataVariablesRep *dv, size\_t offset, Var\_Info *vi)
• static void \texttt{Vchk\_DiscreteStateSetInt} (DataVariablesRep *dv, size\_t offset, Var\_Info *vi)
• static void \texttt{Vchk\_DiscreteStateSetReal} (DataVariablesRep *dv, size\_t offset, Var\_Info *vi)
• static const char * \texttt{Var\_Name} (StringArray *sa, char *buf, size\_t i)
• static void \texttt{Var\_boundchk} (DataVariablesRep *dv, Var\_rcheck *b)
• static void \texttt{Var\_iboundchk} (DataVariablesRep *dv, Var\_icheck *b)
• static void \texttt{flatten\_num\_rva} (RealVectorArray *rva, IntArray **pia)
• static void \texttt{flatten\_num\_rsa} (RealSetArray *rsa, IntArray **pia)
• static void \texttt{flatten\_num\_isa} (IntSetArray *isa, IntArray **pia)
• static void \texttt{flatten\_num\_rrma} (RealRealMapArray *rrma, IntArray **pia)
• static void \texttt{flatten\_num\_irma} (IntRealMapArray *irma, IntArray **pia)
• static void \texttt{flatten\_rva} (RealVectorArray *rva, RealVector **prv)
• static void \texttt{flatten\_iva} (IntVectorArray *iva, IntVector **piv)
• static void \texttt{flatten\_rsm} (RealSymMatrix *rsm, RealVector **prv)
• static void \texttt{flatten\_rsa} (RealSetArray *rsa, RealVector **prv)
• static void \texttt{flatten\_isa} (IntSetArray *isa, IntVector **piv)
• static void \texttt{flatten\_rrma\_keys} (RealRealMapArray *rrma, RealVector **prv)
• static void \texttt{flatten\_rrma\_values} (RealRealMapArray *rrma, RealVector **prv)
• static void \texttt{flatten\_irma\_keys} (IntRealMapArray *irma, IntVector **piv)
• static void \texttt{flatten\_irma\_values} (IntRealMapArray *irma, RealVector **prv)
• static void \texttt{var\_iublv} (const char *keyname, Values *val, VarLabel *vl)
• static Iface\_mp\_Rlit MP3 (failAction, recoveryFnVals, recover)
• static Iface\_mp\_ilil MP3 (failAction, retryLimit, retry)
• static Iface\_mp\_lit MP2 (analysisScheduling, master)
- static Iface_mp_lit MP2 (analysisScheduling, peer)
- static Iface_mp_lit MP2 (evalScheduling, master)
- static Iface_mp_lit MP2 (evalScheduling, peer_dynamic)
- static Iface_mp_lit MP2 (evalScheduling, peer_static)
- static Iface_mp_lit MP2 (failAction, abort)
- static Iface_mp_lit MP2 (failAction, continuation)
- static Iface_mp_lit MP2 (interfaceSynchronization, asynchronous)
- static Iface_mp_lit MP2 (interfaceType, direct)
- static Iface_mp_lit MP2 (interfaceType, fork)
- static Iface_mp_lit MP2 (interfaceType, grid)
- static Iface_mp_lit MP2 (interfaceType, matlab)
- static Iface_mp_lit MP2 (interfaceType, python)
- static Iface_mp_lit MP2 (interfaceType, scilab)
- static Iface_mp_lit MP2 (interfaceType, system)
- static Iface_mp_lit MP2 (asynchLocalEvalScheduling, dynamic)
- static Iface_mp_lit MP2 (asynchLocalEvalScheduling, static)
- 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_ (templateDir)
- static String MP_ (workDir)
- static String2DArray MP_ (analysisComponents)
- static StringArray MP_ (analysisDrivers)
- static StringArray MP_ (templateFiles)
- 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_ (numpyFlag)
- static bool MP_ (restartFileFlag)
- static bool MP_ (templateCopy)
- 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 IntVector MP_(primeBase)
static IntVector MP_(sequenceLeap)
static IntVector MP_(sequenceStart)
static IntVector MP_(stepsPerVariable)
static Method_mp_ilit2 MP3 (replacementType, numberRetained, chc)
static Method_mp_ilit2 MP3 (replacementType, numberRetained, elitist)
static Method_mp_ilit2 MP3 (replacementType, numberRetained, random)
static Method_mp_ilit2z MP3 (crossoverType, numCrossPoints, multi_point_binary)
static Method_mp_ilit2z MP3 (crossoverType, numCrossPoints, multi_point_parameterized_binary)
static Method_mp_ilit2z MP3 (crossoverType, numCrossPoints, multi_point_real)
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 (batchSelectionType, distance_penalty)
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 (evalSynchronize, blocking)
static Method_mp_lit MP2 (evalSynchronize, nonblocking)
static Method_mp_lit MP2 (expansionSampleType, incremental_rhs)
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 (integrationRefine, ais)
static Method_mp_lit MP2 (integrationRefine, is)
static Method_mp_lit MP2 (integrationRefine, mmais)
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 (meritFunction, merit2_squared)
• static Method_mp_lit MP2 (methodName, asynch_pattern_search)
• static Method_mp_lit MP2 (methodName, coliny_beta)
• static Method_mp_lit MP2 (methodName, coliny_cobyla)
• static Method_mp_lit MP2 (methodName, coliny_direct)
• static Method_mp_lit MP2 (methodName, coliny_ea)
• static Method_mp_lit MP2 (methodName, coliny_pattern_search)
• static Method_mp_lit MP2 (methodName, coliny_solis_wets)
• static Method_mp_lit MP2 (methodName, conmin_frcg)
• static Method_mp_lit MP2 (methodName, conmin_mfd)
• static Method_mp_lit MP2 (methodName, dace)
• static Method_mp_lit MP2 (methodName, dot_bfgs)
• static Method_mp_lit MP2 (methodName, dot_frcg)
• static Method_mp_lit MP2 (methodName, dot_mmfd)
• static Method_mp_lit MP2 (methodName, dot_slp)
• static Method_mp_lit MP2 (methodName, dot_sqp)
• static Method_mp_lit MP2 (methodName, efficient_global)
• static Method_mp_lit MP2 (methodName, fsu_cvt)
• static Method_mp_lit MP2 (methodName, fsu_harmonic)
• static Method_mp_lit MP2 (methodName, fsu_hammersley)
• static Method_mp_lit MP2 (methodName, mesh_adaptive_search)
• static Method_mp_lit MP2 (methodName, moga)
• static Method_mp_lit MP2 (methodName, ncsu_direct)
• static Method_mp_lit MP2 (methodName, nl2sol)
• static Method_mp_lit MP2 (methodName, nlplq1_sqp)
• static Method_mp_lit MP2 (methodName, nlssol_sqp)
• static Method_mp_lit MP2 (methodName, nond_adaptive_sampling)
• static Method_mp_lit MP2 (methodName, nond_bayes_calibration)
• static Method_mp_lit MP2 (methodName, nond_efficient_subspace)
• static Method_mp_lit MP2 (methodName, nond_gpais)
• static Method_mp_lit MP2 (methodName, nond_global_evidence)
• static Method_mp_lit MP2 (methodName, nond_global_interval_est)
• static Method_mp_lit MP2 (methodName, nond_global_reliability)
• static Method_mp_lit MP2 (methodName, nond_importance_sampling)
• static Method_mp_lit MP2 (methodName, nond_local_evidence)
• static Method_mp_lit MP2 (methodName, nond_pof_darts)
• static Method_mp_lit MP2 (methodName, nond_polynomial_choiss)
• static Method_mp_lit MP2 (methodName, optpp_cg)
• static Method_mp_lit MP2 (methodName, optpp_fd_newton)
• static Method_mp_lit MP2 (methodName, optpp_g_newton)
- static Method_mp_lit MP2 (methodName, optpp_newton)
- static Method_mp_lit MP2 (methodName, optpp_pds)
- static Method_mp_lit MP2 (methodName, optpp_q_newton)
- static Method_mp_lit MP2 (methodName, psuade_moat)
- static Method_mp_lit MP2 (methodName, richardson_extrap)
- static Method_mp_lit MP2 (methodName, soga)
- static Method_mp_lit MP2 (methodName, surrogate_based_global)
- static Method_mp_lit MP2 (methodName, surrogate_based_local)
- static Method_mp_lit MP2 (methodName, vector_parameter_study)
- static Method_mp_lit MP2 (methodName, list_parameter_study)
- static Method_mp_lit MP2 (methodName, centered_parameter_study)
- static Method_mp_lit MP2 (methodName, multidim_parameter_study)
- static Method_mp_lit MP2 (metropolisType, adaptive)
- static Method_mp_lit MP2 (metropolisType, hastings)
- 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 (nondOptAlgorithm, nip)
- static Method_mp_lit MP2 (nondOptAlgorithm, sqp)
- static Method_mp_lit MP2 (nondOptAlgorithm, ea)
- static Method_mp_lit MP2 (nondOptAlgorithm, ego)
- static Method_mp_lit MP2 (nondOptAlgorithm, lhs)
- static Method_mp_lit MP2 (nondOptAlgorithm, sbo)
- 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 (rejectionType, standard)
- static Method_mp_lit MP2 (rejectionType, delayed)
- static Method_mp_lit MP2 (reliabilityIntegration, first_order)
- static Method_mp_lit MP2 (reliabilityIntegration, second_order)
- static Method_mp_lit MP2 (reliabilitySearchType, amv_plus_u)
- static Method_mp_lit MP2 (reliabilitySearchType, amv_plus_x)
- static Method_mp_lit MP2 (reliabilitySearchType, amv_u)
- static Method_mp_lit MP2 (reliabilitySearchType, amv_x)
- static Method_mp_lit MP2 (reliabilitySearchType, egra_u)
- static Method_mp_lit MP2 (reliabilitySearchType, egra_x)
- static Method_mp_lit MP2 (reliabilitySearchType, no_approx)
- static Method_mp_lit MP2 (reliabilitySearchType, tana_u)
- static Method_mp_lit MP2 (reliabilitySearchType, tana_x)
- 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, num2)
- static Method_mp_lit MP2 (sampleType, incremental_lhs)
- static Method_mp_lit MP2 (sampleType, incremental_random)
- static Method_mp_lit MP2 (sampleType, lhs)
- static Method_mp_lit MP2 (sampleType, random)
- 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 (subMethodName, box_behnken)
- static Method_mp_lit MP2 (subMethodName, central_composite)
- static Method_mp_lit MP2 (subMethodName, dream)
- static Method_mp_lit MP2 (subMethodName, gpmsa)
- static Method_mp_lit MP2 (subMethodName, grid)
- static Method_mp_lit MP2 (subMethodName, lhs)
- static Method_mp_lit MP2 (subMethodName, oa_lhs)
- static Method_mp_lit MP2 (subMethodName, oas)
- static Method_mp_lit MP2 (subMethodName, queso)
- static Method_mp_lit MP2 (subMethodName, random)
- static Method_mp_lit MP2 (subMethodName, converge_order)
- static Method_mp_lit MP2 (subMethodName, converge_qoi)
- static Method_mp_lit MP2 (subMethodName, estimate_order)
- static Method_mp_lit MP2 (trialType, grid)
- static Method_mp_lit MP2 (trialType, halton)
- static Method_mp_lit MP2 (trialType, random)
- static Method_mp_lit2 MP4 (methodName, reliabilitySearchType, nond_local_reliability, "mv")
- static Method_mp_litc MP3 (crossoverType, crossoverRate, null_crossover)
- static Method_mp_litc MP3 (crossoverType, crossoverRate, shuffle_random)
- 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 (postProcessorType, distanceVector, distance_postprocessor)
- static Method_mp_slit MP3 (initializationType, flatFile, flat_file)
- static Method_mp_slit MP3 (methodName, dlDetails, dl_solver)
- static Real MP (absConvTol)
- static Real MP (centeringParam)
- static Real MP (collocationRatio)
- static Real MP (collocRatioTermsOrder)
- static Real MP (constraintPenalty)
- static Real MP (constrPenalty)
- static Real MP (constraintTolerance)
• static Real MP_ (contractFactor)
• static Real MP_ (contractStepLength)
• static Real MP_ (convergenceTolerance)
• static Real MP_ (crossoverRate)
• static Real MP_ (falseConvTol)
• static Real MP_ (functionPrecision)
• static Real MP_ (globalBalanceParam)
• static Real MP_ (gradientTolerance)
• static Real MP_ (grThreshold)
• static Real MP_ (initDelta)
• static Real MP_ (initStepLength)
• static Real MP_ (initTRRadius)
• static Real MP_ (likelihoodScale)
• static Real MP_ (lineSearchTolerance)
• static Real MP_ (localBalanceParam)
• static Real MP_ (maxBoxSize)
• static Real MP_ (maxStep)
• static Real MP_ (minBoxSize)
• static Real MP_ (mutationRate)
• static Real MP_ (mutationScale)
• static Real MP_ (refinementRate)
• static Real MP_ (regressionL2Penalty)
• static Real MP_ (shrinkagePercent)
• static Real MP_ (singConvTol)
• static Real MP_ (singRadius)
• static Real MP_ (smoothFactor)
• static Real MP_ (solnTarget)
• static Real MP_ (stepLenToBoundary)
• static Real MP_ (surrBasedLocalTRContract)
• static Real MP_ (surrBasedLocalTRContractTrigger)
• static Real MP_ (surrBasedLocalTRExpand)
• static Real MP_ (surrBasedLocalTRExpandTrigger)
• static Real MP_ (surrBasedLocalTRInitSize)
• static Real MP_ (surrBasedLocalTRMinSize)
• static Real MP_ (threshDelta)
• static Real MP_ (threshStepLength)
• static Real MP_ (vbdDropTolerance)
• static Real MP_ (volBoxSize)
• static Real MP_ (vns)
• static Real MP_ (xConvTol)
• static RealVector MP_ (anisoDimPref)
• static RealVector MP_ (finalPoint)
• 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_ (listOfPoints)
• static RealVector MP_ (proposalCovScale)
• static RealVector MP_ (regressionNoiseTol)
• static RealVector MP_ (stepVector)
• static RealVectorArray MP_ (genReliabilityLevels)
• static RealVectorArray MP_ (probabilityLevels)
• static RealVectorArray MP_ (reliabilityLevels)
• static RealVectorArray MP_ (responseLevels)
• static unsigned short MP_ (cubIntOrder)
• static unsigned short MP_ (vbdOrder)
• static SizetArray MP_ (collocationPoints)
• static SizetArray MP_ (expansionSamples)
• static UShortArray MP_ (expansionOrder)
• static UShortArray MP_ (quadratureOrder)
• static UShortArray MP_ (sparseGridLevel)
• static UShortArray MP_ (tensorGridOrder)
• static String MP_ (approxExportFile)
• static String MP_ (approxImportFile)
• static String MP_ (betaSolverName)
• static String MP_ (displayFormat)
• static String MP_ (expansionImportFile)
• static String MP_ (historyFile)
• static String MP_ (idMethod)
• static String MP_ (logFile)
• static String MP_ (modelPointer)
• static String MP_ (pstudyFilename)
• static String MP_ (subMethodName)
• static String MP_ (subMethodPointer)
• static StringArray MP_ (linearEqScaleTypes)
• static StringArray MP_ (linearIneqScaleTypes)
• static StringArray MP_ (miscOptions)
• static bool MP_ (approxExportAnnotated)
• static bool MP_ (approxImportAnnotated)
• static bool MP_ (calibrateSigmaFlag)
• static bool MP_ (constantPenalty)
• static bool MP_ (crossValidation)
• static bool MP_ (expansionFlag)
• static bool MP_ (fixedSeedFlag)
• static bool MP_ (fixedSequenceFlag)
• static bool MP_ (latinizeFlag)
• static bool MP_ (mainEffectsFlag)
• static bool MP_ (methodScaling)
- static bool MP_(methodUseDerivsFlag)
- static bool MP_(mutationAdaptive)
- static bool MP_(normalizedCoeffs)
- static bool MP_(printPopFlag)
- static bool MP_(pstudyFileAnnotated)
- static bool MP_(randomizeOrderFlag)
- static bool MP_(regressDiag)
- static bool MP_(showAllEval)
- static bool MP_(showMiscOptions)
- static bool MP_(speculativeFlag)
- static bool MP_(tensorGridFlag)
- static bool MP_(surrBasedGlobalReplacePts)
- static bool MP_(surrBasedLocalLayerBypass)
- static bool MP_(vbdFlag)
- static bool MP_(volQualityFlag)
- static short MP_(expansionType)
- static short MP_(nestingOverride)
- static short MP_(refinementType)
- static int MP_(batchSize)
- static int MP_(contractAfterFail)
- static int MP_(covarianceType)
- static int MP_(crossoverChainPairs)
- static int MP_(emulatorSamples)
- static int MP_(expandAfterSuccess)
- static int MP_(jumpStep)
- static int MP_(maxFunctionEvaluations)
- static int MP_(maxIterations)
- static int MP_(mutationRange)
- 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_(previousSamples)
- static int MP_(randomSeed)
- static int MP_(searchSchemeSize)
- static int MP_(surrBasedLocalSoftConvLimit)
- static int MP_(totalPatternSize)
- static int MP_(verifyLevel)
- 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 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, GAUSSIAN_PROCESS)
- static Method_mp_type MP2s (emulatorType, KRIGING)
- static Method_mp_type MP2s (emulatorType, POLYNOMIAL_CHAOS)
- static Method_mp_type MP2s (emulatorType, STOCHASTIC_COLLOCATION)
- static Method_mp_type MP2s (expansionType, ASKEY_U)
- static Method_mp_type MP2s (expansionType, STD_NORMAL_U)
- static Method_mp_type MP2p (growthOverride, RESTRICTED)
- static Method_mp_type MP2p (growthOverride, UNRESTRICTED)
- static Method_mp_type MP2s (IsRegressionType, EQ_CON_LS)
- static Method_mp_type MP2s (IsRegressionType, 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 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 (refinementType, P_REFINEMENT)
- static Method_mp_type MP2p (refinementType, 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 (sparseGridBasisType, NODAL_INTERPOLANT)
static Method_mp_type MP2s (sparseGridBasisType, HIERARCHICAL_INTERPOLANT)
static Method_mp_type MP2s (surrBasedLocalAcceptLogic, FILTER)
static Method_mp_type MP2s (surrBasedLocalAcceptLogic, TR_RATIO)
static Method_mp_type MP2s (surrBasedLocalConstrRelax, HOMOTOPY)
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 IntSet MP (surrogateFnIndices)
static Model_mp_lit MP2 (approxPointReuse, all)
static Model_mp_lit MP2 (approxPointReuse, none)
static Model_mp_lit MP2 (approxPointReuse, region)
static Model_mp_lit MP2 (marsInterpolation, linear)
static Model_mp_lit MP2 (marsInterpolation, cubic)
static Model_mp_lit MP2 (modelType, nested)
static Model_mp_lit MP2 (modelType, single)
static Model_mp_lit MP2 (modelType, surrogate)
static Model_mp_lit MP2 (surrogateType, hierarchical)
static Model_mp_lit MP2 (surrogateType, global_gaussian)
static Model_mp_lit MP2 (surrogateType, global_kriging)
static Model_mp_lit 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, local_taylor)
static Model_mp_lit MP2 (surrogateType, multipoint_tana)
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 (approxCorrectionOrder, 0)
static Model_mp_ord MP2s (approxCorrectionOrder, 1)
static Model_mp_ord MP2s (approxCorrectionOrder, 2)
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 Real MP_ (annRange)
- static Real MP_ (krigingNugget)
- static Real MP_ (percentFold)
- static RealVector MP_ (krigingCorrelations)
- static RealVector MP_ (primaryRespCoeffs)
- static RealVector MP_ (secondaryRespCoeffs)
- static String MP_ (approxChallengeFile)
- static String MP_ (approxExportModelFile)
- static String MP_ (idModel)
- static String MP_ (interfacePointer)
- static String MP_ (krigingOptMethod)
- static String MP_ (lowFidelityModelPointer)
- static String MP_ (optionalInterfRespPointer)
- static String MP_ (responsesPointer)
- static String MP_ (truthModelPointer)
- static String Array MP_ (primaryVarMaps)
- static String Array MP_ (secondaryVarMaps)
- static String Array MP_ (diagMetrics)
- static bool MP_ (approxChallengeAnnotated)
- static bool MP_ (crossValidateFlag)
- static bool MP_ (hierarchicalTags)
- static bool MP_ (modelUseDerivsFlag)
- static bool MP_ (pressFlag)
- static bool MP_ (pointSelection)
- static short MP_ (annNodes)
- static short MP_ (annRandomWeight)
- static short MP_ (krigingFindNugget)
- static short MP_ (krigingMaxTrials)
- static short MP_ (marsMaxBases)
- static short MP_ (mlsPolyOrder)
- static short MP_ (mlsWeightFunction)
- static short MP_ (rbfBases)
- static short MP_ (rbfMaxPts)
- static short MP_ (rbfMaxSubsets)
- static short MP_ (rbfMinPartitions)
- static int MP_ (numFolds)
- static int MP_ (pointsTotal)
- static IntSet MP_ (idAnalyticGrads)
- static IntSet MP_ (idAnalyticHessians)
- static IntSet MP_ (idNumericalGrads)
- static IntSet MP_ (idNumericalHessians)
• static IntSet MP_ (idQuasiHessians)
• static IntVector MP_ (numReplicates)
• static RealVector MP_ (expConfigVars)
• static RealVector MP_ (expObservations)
• static RealVector MP_ (expStdDeviations)
• static RealVector MP_ (primaryRespFnsWeights)
• static RealVector MP_ (nonlinearEqTargets)
• static RealVector MP_ (nonlinearIneqLowerBnds)
• static RealVector MP_ (nonlinearIneqUpperBnds)
• static RealVector MP_ (fdGradStepSize)
• static RealVector MP_ (fdHessStepSize)
• static RealVector MP_ (primaryRespFnScales)
• static RealVector MP_ (nonlinearEqScales)
• static RealVector MP_ (nonlinearIneqScales)
• static Resp_mp_lit MP2 (gradientType, analytic)
• static Resp_mp_lit MP2 (gradientType, mixed)
• static Resp_mp_lit MP2 (gradientType, none)
• static Resp_mp_lit MP2 (gradientType, numerical)
• static Resp_mp_lit 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_ (expDataFileName)
• 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 bool MP_ (centralHess)
• static bool MP_ (expDataFileAnnotated)
• static bool MP_ (ignoreBounds)
• static size_t MP_(numExpStdDeviations)
• static size_t MP_(numExpConfigVars)
• static size_t MP_(numExperiments)
• 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 Real MP_(hybridLSProb)
• static RealVector MP_(concurrentParameterSets)
• static Strategy_mp_lit MP2(hybridType, collaborative)
• static Strategy_mp_lit MP2(hybridType, embedded)
• static Strategy_mp_lit MP2(hybridType, sequential)
• static Strategy_mp_lit MP2(iteratorScheduling, master)
• static Strategy_mp_lit MP2(iteratorScheduling, peer)
• static Strategy_mp_lit MP2(strategyType, hybrid)
• static Strategy_mp_lit MP2(strategyType, multi_start)
• static Strategy_mp_lit MP2(strategyType, pareto_set)
• static Strategy_mp_lit MP2(strategyType, single_method)
• static String MP_(hybridGlobalMethodPointer)
• static String MP_(hybridLocalMethodPointer)
• static String MP_(methodPointer)
• static String MP_(resultsOutputFile)
• static String MP_(tabularDataFile)
• static StringArray MP_(hybridMethodList)
• static bool MP_(graphicsFlag)
• static bool MP_(resultsOutputFlag)
• static bool MP_(tabularDataFlag)
• static int MP_(concurrentRandomJobs)
• static int MP_(concurrentSeed)
• static int MP_(iteratorServers)
• static int MP_(outputPrecision)
• 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_(numDiscreteDesSetRealVars)
• static size_t MP_(numDiscreteIntervalUncVars)
• static size_t MP_(numDiscreteStateRangeVars)
• static size_t MP_(numDiscreteStateSetIntVars)
• static size_t MP_(numDiscreteStateSetRealVars)
• static size_t MP_(numDiscreteUncSetIntVars)
• static size_t MP_(numDiscreteUncSetRealVars)
• 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_(numHistogramPtUncVars)
• 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 MP_(binomialUncNumTrials)
• static IntVector MP_(discreteDesignRangeLowerBnds)
• static IntVector MP_(discreteDesignRangeUpperBnds)
• static IntVector MP_(discreteDesignRangeVars)
• static IntVector MP_(discreteDesignSetIntVars)
• static IntVector VP_(ddsi)
• static IntVector VP_(dssi)
• static IntVector VP_(dusi)
• static IntVector VP_(DIlb)
• static IntVector VP_(DIlub)
• static IntVector MP_(discreteStateRangeLowerBnds)
• static IntVector MP_(discreteStateRangeUpperBnds)
• static IntVector MP_(discreteStateRangeVars)
• static IntVector MP_(discreteStateSetIntVars)
• static IntVector MP_(hyperGeomUncTotalPop)
• static IntVector MP_(hyperGeomUncSelectedPop)
• static IntVector MP_(hyperGeomUncNumDrawn)
• static IntVector MP_(negBinomialUncNumTrials)
• static IntArray VP_(nddsi)
• static IntArray VP_(nddsr)
• static IntArray VP_(ndssr)
• static IntArray VP_(ndusi)
• static IntArray VP_(ndusr)
• static IntArray VP_(nlpb)
• static IntArray VP_(nhpb)
• static IntArray VP_(nhpp)
• static IntArray VP_(nCI)
• static IntArray VP_(nDI)
• static RealVector MP_(betaUncLowerBnds)
• static RealVector MP_(betaUncUpperBnds)
• static RealVector MP_ (binomialUncProbPerTrial)
• static RealVector MP_ (continuousDesignLowerBnds)
• static RealVector MP_ (continuousDesignUpperBnds)
• static RealVector MP_ (continuousDesignVars)
• static RealVector MP_ (continuousDesignScales)
• static RealVector MP_ (continuousStateLowerBnds)
• static RealVector MP_ (continuousStateUpperBnds)
• static RealVector MP_ (continuousStateVars)
• static RealVector MP_ (discreteDesignSetRealVars)
• static RealVector MP_ (discreteStateSetRealVars)
• static RealVector MP_ (frechetUncBetas)
• static RealVector MP_ (geometricUncProbPerTrial)
• static RealVector MP_ (gumbelUncBetas)
• static RealVector MP_ (negBinomialUncProbPerTrial)
• static RealVector MP_ (normalUncLowerBnds)
• static RealVector MP_ (normalUncMeans)
• static RealVector MP_ (normalUncUpperBnds)
• static RealVector MP_ (poissonUncLambdas)
• static RealVector MP_ (triangularUncModes)
• static RealVector VP_ (ddsr)
• static RealVector VP_ (dssr)
• static RealVector VP_ (dusr)
• static RealVector VP_ (CIIb)
• static RealVector VP_ (CIUb)
• static RealVector VP_ (CIp)
• static RealVector VP_ (DIp)
• static RealVector VP_ (DSIp)
• static RealVector VP_ (DSRp)
• static RealVector VP_ (hba)
• static RealVector VP_ (hbo)
• static RealVector VP_ (hbc)
• static RealVector VP_ (hpa)
• static RealVector VP_ (hpc)
• 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_ (discreteDesignSetRealLabels)
• static StringArray MP_ (discreteStateRangeLabels)
• static StringArray MP_ (discreteStateSetIntLabels)
• static StringArray MP_ (discreteStateSetRealLabels)
• static Var_brv MP2s (betaUncAlphas, 0.)
• static Var_brv MP2s (betaUncBetas, 0.)
- static Var_br_v MP2s (exponentialUncBetas, 0.)
- static Var_br_v MP2s (frechetUncAlphas, 2.)
- static Var_br_v MP2s (gammaUncAlphas, 0.)
- static Var_br_v MP2s (gammaUncBetas, 0.)
- static Var_br_v MP2s (gumbelUncAlphas, 0.)
- static Var_br_v MP2s (lognormalUncErrFacts, 1.)
- static Var_br_v MP2s (lognormalUncLambdas, 0.)
- static Var_br_v MP2s (lognormalUncLowerBnds, 0.)
- static Var_br_v MP2s (lognormalUncMeans, 0.)
- static Var_br_v MP2s (lognormalUncStdDevs, 0.)
- static Var_br_v MP2s (lognormalUncUpperBnds, DBL_MAX)
- static Var_br_v MP2s (lognormalUncZetas, 0.)
- static Var_br_v MP2s (loguniformUncLowerBnds, 0.)
- static Var_br_v MP2s (loguniformUncUpperBnds, DBL_MAX)
- static Var_mp_type Vtype (varsDomain, MIXED_DOMAIN)
- static Var_mp_type Vtype (varsDomain, RELAXED_DOMAIN)
- static Var_mp_type Vtype (varsView, ALL_VIEW)
- static Var_mp_type Vtype (varsView, DESIGN_VIEW)
- static Var_mp_type Vtype (varsView, UNCERTAIN_VIEW)
- static Var_mp_type Vtype (varsView, ALEATORY_UNCERTAIN_VIEW)
- static Var_mp_type Vtype (varsView, EPISTEMIC_UNCERTAIN_VIEW)
- static Var_mp_type Vtype (varsView, STATE_VIEW)
- void dn2f_ (int *n, int *p, Real *x, Calcrj, int *lv, Real *v, int *ui, void *ur, Vf)
- void dn2f_ (int *n, int *p, Real *x, Real *b, Calcrj, int *iv, int *lv, Real *v, int *ui, void *ur, Vf)
- void dn2gb_ (int *n, int *p, Real *x, Real *b, Calcrj, Calcrj, int *iv, 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 (Model &model)
- NLPQLPOptimizer * new_NLPQLPOptimizer (NoDBBaseConstructor, Model &model)
- NPSOLOptimizer * new_NPSOLOptimizer (Model &model)
- NPSOLOptimizer * new_NPSOLOptimizer1 (NoDBBaseConstructor, 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` (NoDBBaseConstructor dummy, Model &model)

- `NPSOLOptimizer * new_NPSOLOptimizer` (Model &model, const int &, const Real &)

- `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`

- `BiStream & operator>>(BiStream &s, ParamResponsePair &pair)`
  
  `BiStream extraction operator for ParamResponsePair`

- `BoStream & operator<<(BoStream &s, const ParamResponsePair &pair)`
  
  `BoStream insertion operator for ParamResponsePair`

- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ParamResponsePair &pair)`
  
  `MPIUnpackBuffer extraction operator for ParamResponsePair`

- `MPIPackBuffer & operator<<(MPIPackBuffer &s, const ParamResponsePair &pair)`
  
  `MPIPackBuffer insertion operator for ParamResponsePair`

- `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)`

- `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.

- **bool lookup_by_val (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr, ParamResponsePair &found_pr)**
  alternate overloaded form returns bool and sets found_pr by wrapping lookup_by_val(PRPMultiIndexCache&, ParamResponsePair&)

- **PRPCacheHIter lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)**
  find the evaluation id of a ParamResponsePair within a PRPMultiIndexCache based on interface id, variables, and ActiveSet search data

- **bool lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, ParamResponsePair &found_pr)**
  find a ParamResponsePair within a PRPMultiIndexCache based on interface id, variables, and ActiveSet search data

- **bool lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, int &found_eval_id)**
  find the evaluation id of a ParamResponsePair within a PRPMultiIndexCache based on interface id, variables, and ActiveSet search data

- **bool lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, Response &found_resp)
find the response of a ParamResponsePair within a PRPMultiIndexCache based on interface id, variables, and ActiveSet search data

- 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

- bool lookup_by_ids (PRPMultiIndexCache &prp_cache, const IntStringPair &search_eval_interface_ids, ParamResponsePair &found_pr)
  find a ParamResponsePair within a PRPMultiIndexCache based on eval_interface_ids

- bool lookup_by_ids (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr, ParamResponsePair &found_pr)
  find a ParamResponsePair within a PRPMultiIndexCache based on eval_interface_ids from the ParamResponsePair search data

- 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.

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const ParamResponsePair &search_pr, ParamResponsePair &found_pr)
  alternate overloaded form returns bool and sets found_pr by wrapping lookup_by_val(PRPMultiIndexQueue&, ParamResponsePair&)

- PRPQueueHIter lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)
  find the evaluation id of a ParamResponsePair within a PRPMultiIndexQueue based on interface id, variables, and ActiveSet search data

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, ParamResponsePair &found_pr)
  find a ParamResponsePair within a PRPMultiIndexQueue based on interface id, variables, and ActiveSet search data

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, int &found_eval_id)
  find the evaluation id of a ParamResponsePair within a PRPMultiIndexQueue based on interface id, variables, and ActiveSet search data

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, Response &found Resp)
  find the response of a ParamResponsePair within a PRPMultiIndexQueue based on interface id, variables, and ActiveSet search data

- PRPQueueOIter lookup_by_eval_id (PRPMultiIndexQueue &prp_queue, const int &search_id)
  find a ParamResponsePair within a PRPMultiIndexQueue based on search_id (i.e. integer eval_id) search data
### 12.1 Dakota Namespace Reference

- bool `lookup_by_eval_id` (PRPMultiIndexQueue &prp_queue, const int &search_id, ParamResponsePair &found_pr)
  
  *find a ParamResponsePair within a PRPMultiIndexQueue based on eval_id*

- bool `lookup_by_eval_id` (PRPMultiIndexQueue &prp_queue, const ParamResponsePair &search_pr, ParamResponsePair &found_pr)
  
  *find a ParamResponsePair within a PRPMultiIndexQueue based on eval_id from the ParamResponsePair search data*

- void `print_restart` (int argc, char **argv, String print_dest)
  
  *print a restart file*

- void `print_restart_tabular` (int argc, char **argv, String print_dest)
  
  *print a restart file (tabular format)*

- void `read_neutral` (int argc, char **argv)
  
  *read a restart file (neutral file format)*

- void `repair_restart` (int argc, char **argv, String identifier_type)
  
  *repair a restart file by removing corrupted evaluations*

- void `concatenate_restart` (int argc, char **argv)
  
  *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)

- int `salinas_main` (int argc, char *argv[], MPI_Comm *comm)
  
  *subroutine interface to SALINAS simulation code*

- void `find_env_token` (const char *s0, const char **s1, const char **s2, const char **s3)

- const char **`arg_list_adjust` (const char **, void **)
  
  *Utility function from legacy, "not_executable" module -- DO NOT TOUCH!*

- bool `contains` (const bfs::path &dir_path, const std::string &file_name, boost::filesystem::path &complete_filepath)
  
  *Helper for "which" - sets complete_filepath from dir_path/file_name combo.*

### Variables

- **BoStream** `write_restart`  
  
  *the restart binary output stream (doesn’t < really need to be global anymore except for < abort_handler()).*

- **PRPCache** `data_pairs`  
  
  *contains all parameter/response pairs.*
• **ParallelLibrary** *Dak_pl*
  set by `ParallelLibrary`, for use in CLH

• static `time_t` **start_time**
• static `int` **dakdrive**
• static `char` **slmap** [256]
• `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.

• **Graphics** **dakota_graphics**
  the global `Dakota::Graphics` object used by `<strategies, models, and approximations`

• `int` **write_precision = 10**
  used in ostream data output functions `<restart_util.cpp overrides this default value`

• **ParallelLibrary** **dummy_lib ("dummy")**
  dummy `ParallelLibrary` object used for `<mandatory reference initialization when `<a real `ParallelLibrary` instance is `< unavailable`

• **ProblemDescDB** **dummy_db**
  dummy `ProblemDescDB` object used for `<mandatory reference initialization when a `<real `ProblemDescDB` instance is unavailable`

• `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

• **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_funcs ∗ DF
• Dakota_funcs DakFuncs0
• const char ∗ FIELD_NAMES []
• const int NUMBER_OF_FIELDS = 23
• static GuiKeyWord kw_1 [3]
• static GuiKeyWord kw_2 [1]
• static GuiKeyWord kw_3 [4]
• static GuiKeyWord kw_4 [1]
• static GuiKeyWord kw_5 [2]
• static GuiKeyWord kw_6 [7]
• static GuiKeyWord kw_7 [8]
• static GuiKeyWord kw_8 [12]
• static GuiKeyWord kw_9 [2]
• static GuiKeyWord kw_10 [2]
• static GuiKeyWord kw_11 [3]
• static GuiKeyWord kw_12 [2]
• static GuiKeyWord kw_13 [2]
• static GuiKeyWord kw_14 [8]
• static GuiKeyWord kw_15 [2]
• static GuiKeyWord kw_16 [1]
• static GuiKeyWord kw_17 [1]
• static GuiKeyWord kw_18 [2]
• static GuiKeyWord kw_19 [4]
• static GuiKeyWord kw_20 [2]
• static GuiKeyWord kw_21 [3]
• static GuiKeyWord kw_22 [2]
• static GuiKeyWord kw_23 [2]
• static GuiKeyWord kw_24 [3]
• static GuiKeyWord kw_25 [2]
• static GuiKeyWord kw_26 [14]
• static GuiKeyWord kw_27 [7]
• static GuiKeyWord kw_28 [2]
• static GuiKeyWord kw_29 [18]
• static GuiKeyWord kw_30 [2]
• static GuiKeyWord kw_31 [2]
• static GuiKeyWord kw_32 [5]
• static GuiKeyWord kw_33 [1]
• static GuiKeyWord kw_34 [1]
• static GuiKeyWord kw_35 [4]
• static GuiKeyWord kw_36 [6]
• static GuiKeyWord kw_37 [2]
• static GuiKeyWord kw_38 [2]
• static GuiKeyWord kw_39 [2]
• static GuiKeyWord kw_40 [2]
• static GuiKeyWord kw_41 [2]
• static GuiKeyWord kw_42 [7]
• static GuiKeyWord kw_43 [2]
• static GuiKeyWord kw_44 [2]
• static GuiKeyWord kw_45 [5]
• static GuiKeyWord kw_46 [1]
• static GuiKeyWord kw_47 [1]
• static GuiKeyWord kw_48 [4]
• static GuiKeyWord kw_49 [5]
• static GuiKeyWord kw_50 [8]
• static GuiKeyWord kw_51 [3]
• static GuiKeyWord kw_52 [6]
• static GuiKeyWord kw_53 [7]
• static GuiKeyWord kw_54 [2]
• static GuiKeyWord kw_55 [11]
• static GuiKeyWord kw_56 [3]
• static GuiKeyWord kw_57 [2]
• static GuiKeyWord kw_58 [3]
• static GuiKeyWord kw_59 [2]
• static GuiKeyWord kw_60 [5]
• static GuiKeyWord kw_61 [4]
• static GuiKeyWord kw_62 [14]
• static GuiKeyWord kw_63 [3]
• static GuiKeyWord kw_64 [2]
• static GuiKeyWord kw_65 [2]
• static GuiKeyWord kw_66 [17]
• static GuiKeyWord kw_67 [13]
• static GuiKeyWord kw_68 [11]
• static GuiKeyWord kw_69 [9]
• static GuiKeyWord kw_70 [1]
• static GuiKeyWord kw_71 [14]
• static GuiKeyWord kw_72 [14]
• static GuiKeyWord kw_73 [2]
• static GuiKeyWord kw_74 [2]
• static GuiKeyWord kw_75 [2]
• static GuiKeyWord kw_76 [6]
• static GuiKeyWord kw_77 [8]
• static GuiKeyWord kw_78 [3]
• static GuiKeyWord kw_79 [1]
• static GuiKeyWord kw_80 [8]
• static GuiKeyWord kw_81 [1]
• static GuiKeyWord kw_82 [10]
• static GuiKeyWord kw_83 [2]
• static GuiKeyWord kw_84 [2]
• static GuiKeyWord kw_85 [2]
• static GuiKeyWord kw_86 [3]
• static GuiKeyWord kw_87 [2]
• static GuiKeyWord kw_88 [10]
• static GuiKeyWord kw_89 [2]
• static GuiKeyWord kw_90 [3]
• static GuiKeyWord kw_91 [2]
• static GuiKeyWord kw_92 [2]
• static GuiKeyWord kw_93 [2]
• static GuiKeyWord kw_94 [2]
• static GuiKeyWord kw_95 [5]
• static GuiKeyWord kw_96 [11]
• static GuiKeyWord kw_97 [2]
• static GuiKeyWord kw_98 [2]
• static GuiKeyWord kw_99 [2]
• static GuiKeyWord kw_100 [2]
• static GuiKeyWord kw_101 [5]
• static GuiKeyWord kw_102 [7]
• static GuiKeyWord kw_103 [2]
• static GuiKeyWord kw_104 [1]
• static GuiKeyWord kw_105 [1]
• static GuiKeyWord kw_106 [2]
• static GuiKeyWord kw_107 [2]
• static GuiKeyWord kw_108 [2]
• static GuiKeyWord kw_109 [3]
• static GuiKeyWord kw_110 [2]
• static GuiKeyWord kw_111 [2]
• static GuiKeyWord kw_112 [2]
• static GuiKeyWord kw_113 [13]
• static GuiKeyWord kw_114 [2]
• static GuiKeyWord kw_115 [3]
• static GuiKeyWord kw_116 [2]
• static GuiKeyWord kw_117 [10]
• static GuiKeyWord kw_118 [2]
• static GuiKeyWord kw_119 [2]
• static GuiKeyWord kw_120 [2]
• static GuiKeyWord kw_121 [1]
• static GuiKeyWord kw_122 [1]
• static GuiKeyWord kw_123 [2]
• static GuiKeyWord kw_124 [3]
• static GuiKeyWord kw_125 [2]
• static GuiKeyWord kw_126 [6]
• static GuiKeyWord kw_127 [2]
• static GuiKeyWord kw_128 [5]
• static GuiKeyWord kw_129 [3]
• static GuiKeyWord kw_130 [10]
• static GuiKeyWord kw_131 [1]
• static GuiKeyWord kw_132 [2]
• static GuiKeyWord kw_133 [4]
• static GuiKeyWord kw_134 [2]
• static GuiKeyWord kw_135 [6]
• static GuiKeyWord kw_136 [15]
• static GuiKeyWord kw_137 [2]
• static GuiKeyWord kw_138 [5]
• static GuiKeyWord kw_139 [3]
• static GuiKeyWord kw_140 [1]
• static GuiKeyWord kw_141 [6]
• static GuiKeyWord kw_142 [3]
• static GuiKeyWord kw_143 [2]
• static GuiKeyWord kw_144 [1]
• static GuiKeyWord kw_145 [3]
• static GuiKeyWord kw_146 [1]
• static GuiKeyWord kw_147 [2]
• static GuiKeyWord kw_148 [4]
• static GuiKeyWord kw_149 [21]
• static GuiKeyWord kw_150 [1]
• static GuiKeyWord kw_151 [4]
• static GuiKeyWord kw_152 [9]
• static GuiKeyWord kw_153 [2]
• static GuiKeyWord kw_154 [3]
• static GuiKeyWord kw_155 [2]
• static GuiKeyWord kw_156 [7]
• static GuiKeyWord kw_157 [1]
• static GuiKeyWord kw_158 [2]
• static GuiKeyWord kw_159 [4]
• static GuiKeyWord kw_160 [2]
• static GuiKeyWord kw_161 [2]
• static GuiKeyWord kw_162 [1]
• static GuiKeyWord kw_163 [1]
• static GuiKeyWord kw_164 [2]
• static GuiKeyWord kw_165 [2]
• static GuiKeyWord kw_166 [1]
• static GuiKeyWord kw_167 [18]
• static GuiKeyWord kw_168 [4]
• static GuiKeyWord kw_169 [5]
• static GuiKeyWord kw_170 [2]
• static GuiKeyWord kw_171 [2]
• static GuiKeyWord kw_172 [6]
• static GuiKeyWord kw_173 [3]
• static GuiKeyWord kw_174 [2]
• static GuiKeyWord kw_175 [3]
• static GuiKeyWord kw_176 [3]
• static GuiKeyWord kw_177 [2]
• static GuiKeyWord kw_178 [5]
• static GuiKeyWord kw_179 [2]
• static GuiKeyWord kw_180 [26]
• static GuiKeyWord kw_181 [1]
• static GuiKeyWord kw_182 [4]
• static GuiKeyWord kw_183 [1]
• static GuiKeyWord kw_184 [11]
• static GuiKeyWord kw_185 [2]
• static GuiKeyWord kw_186 [2]
• static GuiKeyWord kw_187 [3]
• static GuiKeyWord kw_188 [2]
• static GuiKeyWord kw_189 [2]
• static GuiKeyWord kw_190 [3]
• static GuiKeyWord kw_191 [2]
• static GuiKeyWord kw_192 [4]
• static GuiKeyWord kw_193 [2]
• static GuiKeyWord kw_194 [26]
• static GuiKeyWord kw_195 [1]
• static GuiKeyWord kw_196 [12]
• static GuiKeyWord kw_197 [11]
• static GuiKeyWord kw_198 [10]
• static GuiKeyWord kw_199 [3]
• static GuiKeyWord kw_200 [4]
• static GuiKeyWord kw_201 [15]
• static GuiKeyWord kw_202 [5]
• static GuiKeyWord kw_203 [3]
• static GuiKeyWord kw_204 [4]
• static GuiKeyWord kw_205 [2]
• static GuiKeyWord kw_206 [2]
• static GuiKeyWord kw_207 [2]
• static GuiKeyWord kw_208 [2]
• static GuiKeyWord kw_209 [4]
• static GuiKeyWord kw_210 [19]
• static GuiKeyWord kw_211 [14]
• static GuiKeyWord kw_212 [3]
• static GuiKeyWord kw_213 [2]
• static GuiKeyWord kw_214 [7]
• static GuiKeyWord kw_215 [1]
• static GuiKeyWord kw_216 [4]
• static GuiKeyWord kw_217 [6]
• static GuiKeyWord kw_218 [18]
• static GuiKeyWord kw_219 [3]
• static GuiKeyWord kw_220 [86]
• static GuiKeyWord kw_221 [1]
• static GuiKeyWord kw_222 [4]
• static GuiKeyWord kw_223 [2]
• static GuiKeyWord kw_224 [1]
• static GuiKeyWord kw_225 [2]
• static GuiKeyWord kw_226 [6]
• static GuiKeyWord kw_227 [2]
• static GuiKeyWord kw_228 [2]
• static GuiKeyWord kw_229 [2]
• static GuiKeyWord kw_230 [3]
• static GuiKeyWord kw_231 [2]
• static GuiKeyWord kw_232 [4]
• static GuiKeyWord kw_233 [7]
• static GuiKeyWord kw_234 [2]
• static GuiKeyWord kw_235 [2]
• static GuiKeyWord kw_236 [2]
• static GuiKeyWord kw_237 [3]
• static GuiKeyWord kw_238 [3]
• static GuiKeyWord kw_239 [4]
• static GuiKeyWord kw_240 [4]
• static GuiKeyWord kw_241 [5]
• static GuiKeyWord kw_242 [3]
• static GuiKeyWord kw_243 [21]
• static GuiKeyWord kw_244 [6]
• static GuiKeyWord kw_245 [3]
• static GuiKeyWord kw_246 [2]
• static GuiKeyWord kw_247 [2]
• static GuiKeyWord kw_248 [5]
• static GuiKeyWord kw_249 [7]
• static GuiKeyWord kw_250 [6]
• static GuiKeyWord kw_251 [6]
• static GuiKeyWord kw_252 [8]
• static GuiKeyWord kw_253 [15]
• static GuiKeyWord kw_254 [4]
• static GuiKeyWord kw_255 [10]
• static GuiKeyWord kw_256 [2]
• static GuiKeyWord kw_257 [1]
• static GuiKeyWord kw_258 [2]
• static GuiKeyWord kw_259 [8]
• static GuiKeyWord kw_260 [6]
• static GuiKeyWord kw_261 [8]
• static GuiKeyWord kw_262 [11]
• static GuiKeyWord kw_263 [8]
• static GuiKeyWord kw_264 [7]
• static GuiKeyWord kw_265 [1]
• static GuiKeyWord kw_266 [2]
• static GuiKeyWord kw_267 [19]
• static GuiKeyWord kw_268 [1]
• static GuiKeyWord kw_269 [3]
• static GuiKeyWord kw_270 [1]
• static GuiKeyWord kw_271 [5]
• static GuiKeyWord kw_272 [2]
• static GuiKeyWord kw_273 [1]
• static GuiKeyWord kw_274 [3]
• static GuiKeyWord kw_275 [1]
• static GuiKeyWord kw_276 [5]
• static GuiKeyWord kw_277 [1]
• static GuiKeyWord kw_278 [1]
• static GuiKeyWord kw_279 [1]
• static GuiKeyWord kw_280 [10]
• static GuiKeyWord kw_281 [6]
• static GuiKeyWord kw_282 [10]
• static GuiKeyWord kw_283 [4]
• static GuiKeyWord kw_284 [12]
• static GuiKeyWord kw_285 [9]
• static GuiKeyWord kw_286 [8]
• static GuiKeyWord kw_287 [8]
• static GuiKeyWord kw_288 [4]
• static GuiKeyWord kw_289 [4]
• static GuiKeyWord kw_290 [8]
• static GuiKeyWord kw_291 [8]
• static GuiKeyWord kw_292 [4]
• static GuiKeyWord kw_293 [4]
• static GuiKeyWord kw_294 [5]
• static GuiKeyWord kw_295 [5]
• static GuiKeyWord kw_296 [4]
• static GuiKeyWord kw_297 [6]
• static GuiKeyWord kw_298 [6]
• static GuiKeyWord kw_299 [3]
• static GuiKeyWord kw_300 [6]
• static GuiKeyWord kw_301 [10]
• static GuiKeyWord kw_302 [8]
• static GuiKeyWord kw_303 [4]
• static GuiKeyWord kw_304 [2]
• static GuiKeyWord kw_305 [4]
• static GuiKeyWord kw_306 [10]
• static GuiKeyWord kw_307 [6]
• static GuiKeyWord kw_308 [4]
• static GuiKeyWord kw_309 [10]
• static GuiKeyWord kw_310 [2]
- static GuiKeyWord kw_311 [8]
- static GuiKeyWord kw_312 [6]
- static GuiKeyWord kw_313 [6]
- static GuiKeyWord kw_314 [37]
- static GuiKeyWord kw_315 [6]
- static KeyWord kw_316 [10]
- static KeyWord kw_317 [8]
- static KeyWord kw_318 [4]
- static KeyWord kw_319 [2]
- static KeyWord kw_320 [4]
- static KeyWord kw_321 [10]
- static KeyWord kw_322 [6]
- static KeyWord kw_323 [4]
- static KeyWord kw_324 [10]
- static KeyWord kw_325 [2]
- static KeyWord kw_326 [8]
- static KeyWord kw_327 [6]
- static KeyWord kw_328 [6]
- static KeyWord kw_329 [37]
- static KeyWord kw_330 [6]
- FILE * nidrin
- static const char * aln_scaletypes [ ] = { "auto", "log", "none", 0 }
- static Var_uinfo CAUVLbl [CAUVar_Nkinds]
- static Var_uinfo DAUIVLbl [DAUIVar_Nkinds]
- static Var_uinfo DAURVLbl [DAURVar_Nkinds]
- static Var_uinfo CEUVLbl [CEUVar_Nkinds]
- static Var_uinfo DEUIVLbl [DEUVar_Nkinds]
- static Var_uinfo DEURVLbl [DEURVar_Nkinds]
- static Var_uinfo DiscSetLbl [DiscSetVar_Nkinds]
- static VarLabelChk Vlch [ ]
- static VLreal VLR [N_VLR]
- static VLint VLI [N_VLI]
- static int VLR_aleatory [N_VLR] = { 1, 0, 1, 0 }
- static int VLI_aleatory [N_VLI] = { 1, 0 }
- 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_dau [ ]
- static Var_check var_mp_check_daur [ ]
- static Var_check var_mp_check_ceu [ ]
- static Var_check var_mp_check_deu [ ]
- static Var_check var_mp_check_deur [ ]
- static Var_rcheck var_mp_cbound [ ]
- static Var_ichck var_mp_drange [ ]
- const char * SCI_FIELD_NAMES [ ]
- const int SCI_NUMBER_OF_FIELDS = 26
12.1 Dakota Namespace Reference

- 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 textbook exponent: 1.0 is nominal, 1.4 used for B&B testing

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 *.CH.

12.1.2 Typedef Documentation

12.1.2.1 typedef bmi::multi_index_container

```c++
<
Dakota::ParamResponsePair,
    bmi::indexed_by<
bmi::ordered_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. For a global cache, both evaluation and interface id’s are used for tagging ParamResponsePair records.

12.1.2.2 typedef bmi::multi_index_container

```c++
<
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. 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 Function Documentation

12.1.3.1 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.

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().
12.1.3.2  void start_dakota_heartbeat (int)
Heartbeat function provided by not_executable.C; pass output interval in seconds, or -1 to use $DAKOTA_-HEARTBEAT
Referenced by ParallelLibrary::init_mpi_comm().

12.1.3.3  int Dakota::my_cp (const char *file, const struct stat *sb, int ftype, int depth, void *v)
my_cp is a wrapper around 'cp -r'. The extra layer allows for symlink to be used instead of file copy.

12.1.3.4  void Dakota::get_npath (int appdrive, std::string *pnpath)
get_npath "shuffles" the string representing the current $PATH variable definition so that "." is first in the $PATH. It then returns the new string as the result (last arg in the call).
Referenced get_cwd().

12.1.3.5  std::string get_cwd ()
Portability adapter for getcwd. Portability adapter for getcwd: return the string in OS-native format. TODO: change paths throughout code to use bfs::path where possible, since Windows (and Cygwin) use wchar_t instead of char_t.
Referenced by get_npath().

12.1.3.6  T Dakota::abort_handler_t (int code) [inline]
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().

12.1.3.7  Real Dakota::getdist (const RealVector &x1, const RealVector &x2)
Gets the Euclidean distance between x1 and x2
Referenced by mindist(), and mindistindx().

12.1.3.8  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().
12.1 Dakota Namespace Reference

12.1.3.9  **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().

12.1.3.10  **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().

12.1.3.11  **int Dakota::start_grid_computing (char * analysis_driver_script, char * params_file, char * results_file)**

sample function prototype for launching grid computing

12.1.3.12  **int Dakota::stop_grid_computing ()**

sample function prototype for terminating grid computing

12.1.3.13  **int Dakota::perform_analysis (char * iteration_num)**

sample function prototype for submitting a grid evaluation

12.1.3.14  **string Dakota::asstring (const T & val)  [inline]**

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().
12.1.3.15  void run_dakota_data ()

Function to encapsulate the DAKOTA object instantiations for mode 2: direct Data class instantiation. 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 ProblemDescDB::broadcast(), ProblemDescDB::check_input(), DataInterface::dataIfaceRep, DataMethod::dataMethodRep, DataResponses::dataRespRep, DataVariables::dataVarsRep, DataResponsesRep::gradientType, DataResponsesRep::hessianType, ProblemDescDB::insert_node(), ProblemDescDB::lock(), DataMethodRep::methodName, model_interface_plugins(), ParallelLibrary::mpirun_flag(), DataVariablesRep::numContinuousDesVars, DataResponsesRep::numNonlinearIneqConstraints, DataResponsesRep::numObjectiveFunctions, ProblemDescDB::post_process(), Strategy::run_strategy(), and ParallelLibrary::world_rank().

Referenced by main().

12.1.3.16  bool Dakota::set_compare (const ParamResponsePair & database_pr, const ActiveSet & search_set) [inline]

search function for a particular ParamResponsePair within a PRPList based on ActiveSet content (request vector and derivative variables vector) a global function to compare the ActiveSet of a particular database_pr (presumed to be in the global history list) with a passed in ActiveSet (search_set).

References ParamResponsePair::active_set(), ActiveSet::derivative_vector(), and ActiveSet::request_vector().

Referenced by lookup_by_val().

12.1.3.17  bool Dakota::id_vars_exact_compare (const ParamResponsePair & database_pr, const ParamResponsePair & search_pr) [inline]

search function for a particular 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 in key of interface id and variables provided by search_pr.

References binary_equal_to(), ParamResponsePair::interface_id(), and ParamResponsePair::prp_parameters().

Referenced by partial_prp_equality::operator().

12.1.3.18  PRPCacheHIter Dakota::lookup_by_val (PRPMultiIndexCache & prp_cache, const ParamResponsePair & search_pr) [inline]

find a ParamResponsePair based on the interface id, variables, and 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 ActiveSet subset logic.

References ParamResponsePair::active_set(), and set_compare().

Referenced by ApplicationInterface::duplication_detect(), Model::estimate_derivatives(), SurrBasedLocalMinimizer::find_center_approx(), Optimizer::local_objective_recast_retrieve(), lookup_by_val(), SNLLLeastSq::post_run(), SurrBasedMinimizer::print_results(), Optimizer::print_results(), LeastSq::print_results(), DiscrepancyCorrection::search_db(), and NonDLocalReliability::update_mpp_search_data().
12.1 Dakota Namespace Reference

12.1.3.19 PRPQueueHit\ter Dakota::lookup_by\_val (PRPMultiIndexQueue & prp\_queue, const ParamResponsePair & search\_pr) \[inline\]

find a ParamResponsePair based on the interface id, variables, and 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().

12.1.3.20 void print\_restart (int argc, char ** argv, String print\_dest)

print a restart file \texttt{Usage: }"dakota\_restart\_util print dakota.rst"
"dakota\_restart\_util to\_neutral dakota.rst dakota.neu"

Prints all evals. 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 ParamResponsePair::eval\_id(), ParamResponsePair::write\_annotated(), and write\_precision.

Referenced by main().

12.1.3.21 void print\_restart\_tabular (int argc, char ** argv, String print\_dest)

print a restart file (tabular format) \texttt{Usage: }"dakota\_restart\_util to\_pdb dakota.rst dakota.pdb"
"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 Variables::continuous\_variables(), data\_pairs, Variables::discrete\_int\_variables(), Variables::discrete\_real\_variables(), ParamResponsePair::eval\_id(), and Response::function\_values().

Referenced by main().

12.1.3.22 void read\_neutral (int argc, char ** argv)

read a restart file (neutral file format) \texttt{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 ParamResponsePair::read\_annotated(), and write\_restart.

Referenced by main().

12.1.3.23 void repair\_restart (int argc, char ** argv, String identifier\_type)

repair a restart file by removing corrupted evaluations \texttt{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 Response::active_set_request_vector(), contains(), ParamResponsePair::eval_id(), Response::function_values(), ParamResponsePair::prp_response(), and write_restart.

Referenced by main().

12.1.3.24 void concatenate_restart (int argc, char ** argv)

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 write_restart.

Referenced by main().

12.1.4 Variable Documentation

12.1.4.1 char slmap[256] [static]

Initial value:

```
{
0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc, 0xd, 0xe, 0xf,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d,
0x1e, 0x1f,
0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d,
0x2e, 0x2f,
0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d,
0x3e, 0x3f,
0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d,
0x4e, 0x4f,
0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5a, 0x5b, '/', '0',
0x5c, 0x5d, 0x5e, 0x5f,
0x60, 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 0x7b, 0x7c, 0x7d,
0x7e, 0x7f,
0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x8b, 0x8c,
0x8d, 0x8e, 0x8f,
0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9a, 0x9b, 0x9c,
0x9d, 0x9e, 0x9f,
0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xab, 0xac,
0xad, 0xae, 0xaf,
0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xbb, 0xbc,
0xbd, 0xbe, 0xbf,
0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcc,
0xcd, 0xce, 0xcf,
0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xdb, 0xdc,
0xdd, 0xde, 0xdf,
0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea, 0xeb, 0xec,
0xed, 0xee, 0xef,
```
12.1 Dakota Namespace Reference

0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff

12.1.4.2 Dakota_funcs DakFuncs0

Initial value:

```
| fprintf, abort_handler, dlao1ver_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_constraint_targets1, linear_ineq_constraint_coeffs1, linear_eq_constraint_coeffs1, ComputeResponses1, GetFuncs1, GetGrads1, GetContVars1, SetBestContVars1, SetBestDiscVars1, SetBestRespFns1, Get_Real1, Get_Int1, Get_Bool1 |
```

12.1.4.3 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

Referenced by MatlabInterface::matlab_engine_run(), and MatlabInterface::MatlabInterface().

12.1.4.4 const int NUMBER_OF_FIELDS = 23

number of fields in above structure

Referenced by MatlabInterface::matlab_engine_run(), and MatlabInterface::MatlabInterface().
12.1.4.5  static KeyWord kw_1  [static]

Initial value:

```c
{
    "active_set_vector", 8, 0, 1, 0, 2151,
    "evaluation_cache", 8, 0, 2, 0, 2153,
    "restart_file", 8, 0, 3, 0, 2155
}
```

1175 distinct keywords (plus 167 aliases)

12.1.4.6  static KeyWord kw_2  [static]

Initial value:

```c
{
    "processors_per_analysis", 0x19, 0, 1, 0, 2127, 0, 0, 0, 0, 0,"{Number of processors per analysis} InterfCommands.html#InterfApplicDF"
}
```

12.1.4.7  static KeyWord kw_3  [static]

Initial value:

```c
{
    "abort", 8, 0, 1, 1, 2141, 0, 0, 0, 0, 0,"@[CHOOSE failure mitigation]"
}
```

12.1.4.8  static KeyWord kw_4  [static]

Initial value:

```c
{
    "numpy", 8, 0, 1, 0, 2133, 0, 0, 0, 0, 0,"{Python NumPy dataflow} InterfCommands.html#InterfApplicMSP"
}
```

12.1.4.9  static KeyWord kw_5  [static]

Initial value:

```c
{
    "copy", 8, 0, 1, 0, 2121, 0, 0, 0, 0, 0,"{Copy template files} InterfCommands.html#InterfApplicCopy",
    "replace", 8, 0, 2, 0, 2123, 0, 0, 0, 0, 0,"{Replace existing files} InterfCommands.html#InterfApplicReplace"
}
```
12.1.4.10 static KeyWord kw_6 [static]

Initial value:

```c
{
  {"dir_save",0,0,3,0,2114},
  {"dir_tag",0,0,2,0,2112},
  {"directory_save",8,0,3,0,2115,0,0,0,0,0,"{Save work directory}
   InterfCommands.html#InterfApplicF"},
  {"directory_tag",8,0,2,0,2113,0,0,0,0,0,"{Tag work directory}
   InterfCommands.html#InterfApplicF"},
  {"named",11,0,1,0,2111,0,0,0,0,0,"{Name of work directory} Int
   erfCommands.html#InterfApplicF"},
  {"template_directory",11,2,4,0,2117,kw_5,0,0,0,0,0,"{Template di
   rectory} InterfCommands.html#InterfApplicF"},
  {"template_files",15,2,4,0,2119,kw_5,0,0,0,0,0,"{Template files}
   InterfCommands.html#InterfApplicF"}
}
```

12.1.4.11 static KeyWord kw_7 [static]

Initial value:

```c
{
  {"allow_existing_results",8,0,3,0,2099,0,0,0,0,0,"{Allow exist
    ing results files} InterfCommands.html#InterfApplicF"},
  {"aprepro",8,0,5,0,2103,0,0,0,0,0,"{Aprepro parameters file fo
    rmat} InterfCommands.html#InterfApplicF"},
  {"file_save",8,0,7,0,2107,0,0,0,0,0,"{Parameters and results f
    ile saving} InterfCommands.html#InterfApplicF"},
  {"file_tag",8,0,6,0,2105,0,0,0,0,0,"{Parameters and results fi
    le tagging} InterfCommands.html#InterfApplicF"},
  {"parameters_file",11,0,1,0,2095,0,0,0,0,0,"{Parameters file n
    ame} InterfCommands.html#InterfApplicF"},
  {"results_file",11,0,2,0,2097,0,0,0,0,0,"{Results file name} I
    nterfCommands.html#InterfApplicF"},
  {"verbatim",8,0,4,0,2101,0,0,0,0,0,"{Verbatim driver/filter in
    vocation syntax} InterfCommands.html#InterfApplicF"},
  {"work_directory",8,7,8,0,2109,kw_6,0,0,0,0,0,"{Create work dire
    ctory} InterfCommands.html#InterfApplicF"}
}
```

12.1.4.12 static KeyWord kw_8 [static]

Initial value:

```c
{
  {"analysis_components",15,0,1,0,2085,0,0,0,0,0,"{Additional id
    entifiers for use by the analysis_drivers} InterfCommands.html#InterfApplicF"},
  {"deactivate",8,3,6,0,2149,kw_1,0,0,0,0,0,"{Feature deactivation
    } InterfCommands.html#InterfApplicF"},
  {"direct",8,1,4,1,2125,kw_2,0,0,0,0,0,"{CHOOSE interface type}{D
    irect function interface } InterfCommands.html#InterfApplicDF"},
  {"failure_capture",8,4,5,0,2139,kw_3,0,0,0,0,0,"{Failure capturi
    ng} InterfCommands.html#InterfApplicA"},
  {"fork",8,8,4,1,2093,kw_7,0,0,0,0,0,"{Fork interface } InterfCo
```
mmans.html#InterfApplicF",
{"grid",8,0,4,1,2137,0,0,0,0,"{Grid interface} InterfCommands.html#InterfApplicG"},
{"input_filter",11,0,2,0,2087,0,0,0,0,"{Input filter} InterfCommands.html#InterfApplic"},
{"matlab",8,0,4,1,2129,0,0,0,0,"{Matlab interface} InterfCommands.html#InterfApplicMSP"},
{"output_filter",11,0,3,0,2089,0,0,0,0,"{Output filter} InterfCommands.html#InterfApplic"},
{"python",8,1,4,1,2131,kw_4,0.,0.,0.,0,"{Python interface} InterfCommands.html#InterfApplicMSP"},
{"scilab",8,0,4,1,2135,0,0,0,0,"{Scilab interface} InterfCommands.html#InterfApplicMSP"},
{"system",8,8,4,1,2091,kw_7}
)

12.1.4.13 static KeyWord kw_9 [static]

Initial value:

{
{"master",8,0,1,1,2185},
{"peer",8,0,1,1,2187}
}

12.1.4.14 static KeyWord kw_10 [static]

Initial value:

{
{"dynamic",8,0,1,1,2163},
{"static",8,0,1,1,2165}
}

12.1.4.15 static KeyWord kw_11 [static]

Initial value:

{
{"analysis_concurrency",0x19,0,3,0,2167,0,0,0,0,"{Asynchronous analysis concurrency} InterfCommands.html#InterfIndControl"},
{"evaluation_concurrency",0x19,0,1,0,2159,0,0,0,0,"{Asynchronous evaluation concurrency} InterfCommands.html#InterfIndControl"},
{"local_evaluation_scheduling",8,2,2,0,2161,kw_10,0,0,0,0,"{Local evaluation scheduling} InterfCommands.html#InterfIndControl"}
}

12.1.4.16 static KeyWord kw_12 () [static]

Initial value:
12.1.4.17 static KeyWord kw_13 () [static]

Initial value:

```json
{"master",8,0,1,1,2173},
{"peer",8,2,1,1,2175,kw_12,0.,0.,0.,0,"{Peer scheduling of evaluations} InterfCommands.html#InterfIndControl"} }
```

12.1.4.18 static KeyWord kw_14 () [static]

Initial value:

```json
{"algebraic_mappings",11,0,2,0,2081,0,0.,0.,0.,0,"{Algebraic mappings file} InterfCommands.html#InterfAlgebraic"},
{"analysis_drivers",15,12,3,0,2083,kw_8,0.,0.,0.,0,"{Analysis drivers} InterfCommands.html#InterfApplic"},
{"analysis_scheduling",8,2,8,0,2183,kw_9,0.,0.,0.,0,"{Message passing configuration for scheduling of analyses} InterfCommands.html#InterfIndControl"},
{"evaluation_scheduling",8,2,6,0,2171,kw_13,0.,0.,0.,0,"{Message passing configuration for scheduling of evaluations} InterfCommands.html#InterfIndControl"},
{"analysis_servers",0x19,0,7,0,2181,0,0.,0.,0.,0,"{Number of analysis servers} InterfCommands.html#InterfIndControl"},
{"asynchronous",8,3,4,0,2157,kw_11,0.,0.,0.,0,"{Asynchronous interface usage} InterfCommands.html#InterfIndControl"},
{"evaluation_servers",0x19,0,5,0,2169,0,0.,0.,0,"{Number of evaluation servers} InterfCommands.html#InterfIndControl"},
{"id_interface",11,0,1,0,2079,0,0.,0.,0.,0,"{Interface set identifier} InterfCommands.html#InterfIndControl"} }
```

12.1.4.19 static KeyWord kw_15 () [static]

Initial value:

```json
{"complementary",8,0,1,1,1079},
{"cumulative",8,0,1,1,1077} }
```

12.1.4.20 static KeyWord kw_16 () [static]

Initial value:
12.1.4.21 static KeyWord kw_17 () [static]

Initial value:

{  
  {"num_gen_reliability_levels",13,0,1,0,1087,0,0.,0.,0.,0,"{Number of generalized reliability levels} MethodCommands.html#MethodNonD"}  
}

12.1.4.22 static KeyWord kw_18 () [static]

Initial value:

{  
  {"num_probability_levels",13,0,1,0,1083,0,0.,0.,0.,0,"{Number of probability levels} MethodCommands.html#MethodNonD"}  
}

12.1.4.23 static KeyWord kw_19 () [static]

Initial value:

{  
  {"mt19937",8,0,1,1,1091},  
  {"rnum2",8,0,1,1,1093}  
}

12.1.4.24 static KeyWord kw_20 [static]

Initial value:

{  
  {"constant_liar",8,0,1,1,971},  
  {"distance_penalty",8,0,1,1,967},  
  {"naive",8,0,1,1,965},  
  {"topology",8,0,1,1,969}  
}

12.1.4.25 static KeyWord kw_21 [static]

Initial value:

{  
  {"annotated",8,0,1,0,983},  
  {"freeform",8,0,1,0,985}  
}
12.1 Dakota Namespace Reference

<table>
<thead>
<tr>
<th>Function</th>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kw_22 ()</td>
<td>static</td>
</tr>
<tr>
<td></td>
<td>kw_23 ()</td>
<td>static</td>
</tr>
<tr>
<td></td>
<td>kw_24 ()</td>
<td>static</td>
</tr>
<tr>
<td></td>
<td>kw_25</td>
<td>static</td>
</tr>
<tr>
<td></td>
<td>kw_26</td>
<td>static</td>
</tr>
</tbody>
</table>

Initial value:

```c
// Dakota Namespace Reference

12.1.4.26 static KeyWord kw_22 () [static]

Initial value:

```c
{ "annotated", 8, 0, 1, 0, 977 },
{ "freeform", 8, 0, 1, 0, 979 }
```

12.1.4.27 static KeyWord kw_23 () [static]

Initial value:

```c
{ "parallel", 8, 0, 1, 1, 1001 },
{ "series", 8, 0, 1, 1, 999 }
```

12.1.4.28 static KeyWord kw_24 () [static]

Initial value:

```c
{ "gen_reliabilities", 8, 0, 1, 1, 995 },
{ "probabilities", 8, 0, 1, 1, 993 },
{ "system", 8, 2, 2, 0, 997, kw_23 }
```

12.1.4.29 static KeyWord kw_25 [static]

Initial value:

```c
{ "compute", 8, 3, 2, 0, 991, kw_24 },
{ "num_response_levels", 13, 0, 1, 0, 989 }
```

12.1.4.30 static KeyWord kw_26 [static]

Initial value:

```c
// Dakota Namespace Reference

Dakota Version 5.4 Developers Manual generated on November 7, 2013
```
Namespace Documentation

{          
  {"batch_selection", 8, 4, 3, 0, 963, kw_19, 0, 0, 0, 0, "\{Batch selection strategy\} MethodCommands.html#MethodNonDAdaptive"},
  {"batch_size", 9, 0, 4, 0, 973, 0, 0, 0, 0, 0, "\{Batch size (number of points added each iteration)\} MethodCommands.html#MethodNonDAdaptive"},
  {"distribution", 8, 2, 11, 0, 1075, kw_15, 0, 0, 0, 0, "\{Distribution type\} MethodCommands.html#MethodNonD"},
  {"emulator_samples", 9, 0, 1, 0, 953, 0, 0, 0, 0, 0, "\{Number of samples on the emulator to generate a new true sample each iteration\} MethodCommands.html#MethodNonDAdaptive"},
  {"export_points_file", 11, 2, 6, 0, 981, kw_20, 0, 0, 0, 0, 0, "\{File name for exporting approximation-based samples from evaluating the GP\} MethodCommands.html#MethodNonDAdaptive"},
  {"fitness_metric", 8, 3, 2, 0, 955, kw_21, 0, 0, 0, 0, 0, "\{Fitness metric\} MethodCommands.html#MethodNonDAdaptive"},
  {"gen_reliability_levels", 14, 1, 13, 0, 1085, kw_16, 0, 0, 0, 0, 0, "\{Generalized reliability levels\} MethodCommands.html#MethodNonD"},
  {"import_points_file", 11, 2, 5, 0, 975, kw_22, 0, 0, 0, 0, 0, "\{File name for points to be imported as the basis for the initial GP\} MethodCommands.html#MethodNonDAdaptive"},
  {"misc_options", 15, 0, 8, 0, 1003},
  {"probability_levels", 14, 1, 12, 0, 1081, kw_17, 0, 0, 0, 0, 0, "\{Probability levels\} MethodCommands.html#MethodNonD"},
  {"response_levels", 14, 2, 7, 0, 987, kw_25},
  {"rng", 8, 2, 14, 0, 1089, kw_18, 0, 0, 0, 0, 0, "\{Random number generator\} MethodCommands.html#MethodNonDMC"},
  {"samples", 9, 0, 10, 0, 1303, 0, 0, 0, 0, 0, 0, "\{Number of samples\} MethodCommands.html#MethodNonDMC"},
  {"seed", 0x19, 0, 9, 0, 1305, 0, 0, 0, 0, 0, 0, "\{Random seed\} MethodCommands.html#MethodEG"}
}

12.1.4.31 static KeyWord kw_27 () [static]
Initial value:

{          
  {"merit1", 8, 0, 1, 1, 271, 0, 0, 0, 0, 0, 0, "\{THOOSE merit function\}"},
  {"merit1_smooth", 8, 0, 1, 1, 273},
  {"merit2", 8, 0, 1, 1, 275},
  {"merit2_smooth", 8, 0, 1, 1, 277, 0, 0, 0, 0, 0, 0, "@"},
  {"merit_max", 8, 0, 1, 1, 279},
  {"merit_max_smooth", 8, 0, 1, 1, 267},
  {"merit_max_smooth", 8, 0, 1, 1, 269}
}

12.1.4.32 static KeyWord kw_28 () [static]
Initial value:

{          
  {"blocking", 8, 0, 1, 1, 261, 0, 0, 0, 0, 0, 0, "\{THOOSE synchronization\}"},
  {"nonblocking", 8, 0, 1, 1, 263, 0, 0, 0, 0, 0, 0, "@"}
}
### 12.1.4.33 static KeyWord kw_29 () [static]

**Initial value:**

```c
{
    "constraint_penalty",10,0,7,0,281,0,0,0,0,0,"{Constraint penalty} MethodCommands.html#MethodAPPSDC",
    "contraction_factor",10,0,2,0,253,0,0,0,0,0,"{Pattern contraction factor} MethodCommands.html#MethodAPPSDC",
    "initial_delta",10,0,1,0,251,0,0,0,0,0,"{Initial offset value} MethodCommands.html#MethodAPPSDC",
    "linear_equality_constraint_matrix",14,0,14,0,431,0,0,0,0,0,"{Linear equality constraint matrix} MethodCommands.html#MethodIndControl",
    "linear_equality_scale_types",14,0,15,0,435,0,0,0,0,0,"{Linear equality scaling types} MethodCommands.html#MethodIndControl",
    "linear_equality_scales",14,0,17,0,437,0,0,0,0,0,"{Linear equality scales} MethodCommands.html#MethodIndControl",
    "linear_equality_targets",14,0,15,0,433,0,0,0,0,0,"{Linear equality targets} MethodCommands.html#MethodIndControl",
    "linear_inequality_constraint_matrix",14,0,9,0,421,0,0,0,0,0,"{Linear inequality constraint matrix} MethodCommands.html#MethodIndControl",
    "linear_inequality_lower_bounds",14,0,10,0,423,0,0,0,0,0,"{Linear inequality lower bounds} MethodCommands.html#MethodIndControl",
    "linear_inequality_scale_types",14,0,12,0,427,0,0,0,0,0,"{Linear inequality scaling types} MethodCommands.html#MethodIndControl",
    "linear_inequality_scales",14,0,10,0,429,0,0,0,0,0,"{Linear inequality scales} MethodCommands.html#MethodIndControl",
    "linear_inequality_targets",14,0,11,0,425,0,0,0,0,0,"{Linear inequality targets} MethodCommands.html#MethodIndControl",
    "merit_function",8,7,6,0,265,kw_27,0,0,0,0,"{Merit function} MethodCommands.html#MethodAPPSDC",
    "smoothing_factor",10,0,8,0,283,0,0,0,0,0,"{Smoothing factor} MethodCommands.html#MethodAPPSDC",
    "solution_accuracy",2,0,4,0,256,
    "solution_target",10,0,4,0,257,0,0,0,0,0,"{Solution target} MethodCommands.html#MethodAPPSDC",
    "synchronization",8,2,5,0,259,kw_28,0,0,0,0,0,"{Evaluation synchronization} MethodCommands.html#MethodAPPSDC",
    "threshold_delta",10,0,3,0,255,0,0,0,0,0,"{Threshold for offset values} MethodCommands.html#MethodAPPSDC"
}
```

### 12.1.4.34 static KeyWord kw_30 () [static]

**Initial value:**

```c
{
    "annotated",8,0,1,0,12311,
    "freeform",8,0,1,0,1233
}
```

### 12.1.4.35 static KeyWord kw_31 () [static]

**Initial value:**

```c
{
}
```
12.1.4.36 static KeyWord kw_32 () [static]

Initial value:

```
{
    "dakota", 8, 0, 1, 1219,
    "emulator_samples", 9, 0, 2, 0, 1221,
    "export_points_file", 11, 2, 4, 0, 1229, kw_30,
    "import_points_file", 11, 2, 3, 0, 1223, kw_31,
    "surfpack", 8, 0, 1, 1, 1217
}
```

12.1.4.37 static KeyWord kw_33 () [static]

Initial value:

```
{
    "sparse_grid_level", 13, 0, 1, 0, 1237
}
```

12.1.4.38 static KeyWord kw_34 () [static]

Initial value:

```
{
    "sparse_grid_level", 13, 0, 1, 0, 1241
}
```

12.1.4.39 static KeyWord kw_35 () [static]

Initial value:

```
{
    "gaussian_process", 8, 5, 1, 1, 1215, kw_32,
    "kriging", 0, 5, 1, 1, 1214, kw_32,
    "pce", 8, 1, 1, 1, 1235, kw_33,
    "sc", 8, 1, 1, 1, 1239, kw_34
}
```

12.1.4.40 static KeyWord kw_36 [static]

Initial value:
12.1 Dakota Namespace Reference

{  
  {"chains",0x29,0,1,0,1203,0,3.,0.,0.,0,"{Number of chains} Method Commands.html#MethodNonDBayesCalib"},  
  {"crossover_chain_pairs",0x29,0,3,0,1207,0,0.,0.,0.,0,"{Number of chain pairs used in crossover} MethodCommands.html#MethodNonDBayesCalib"},  
  {"emulator",8,4,6,0,1213,kw_35},  
  {"gr_threshold",0x1a,0,4,0,1209,0,0.,0.,0.,0,"{Gelman-Rubin Threshold for convergence} MethodCommands.html#MethodNonDBayesCalib"},  
  {"jump_step",0x29,0,5,0,1211,0,0.,0.,0.,0,"{Jump-Step} MethodCommands.html#MethodNonDBayesCalib"},  
  {"num_cr",0x29,0,2,0,1205,0,1.,0.,0.,0,"{Number of candidate points used in burn-in adaptation} MethodCommands.html#MethodNonDBayesCalib"}
}

12.1.4.41 static KeyWord kw_37 [static]
Initial value:

{  
  {"adaptive",8,0,1,1,1191},  
  {"hastings",8,0,1,1,1189}
}

12.1.4.42 static KeyWord kw_38 () [static]
Initial value:

{  
  {"delayed",8,0,1,1,1185},  
  {"standard",8,0,1,1,1183}
}

12.1.4.43 static KeyWord kw_39 () [static]
Initial value:

{  
  {"mt19937",8,0,1,1,1195},  
  {"rnum2",8,0,1,1,1197}
}

12.1.4.44 static KeyWord kw_40 () [static]
Initial value:

{  
  {"annotated",8,0,1,0,1177},  
  {"freeform",8,0,1,0,1179}
}
12.1.4.45 static KeyWord kw_41 () [static]

Initial value:

```cpp
{
  {"annotated",8,0,1,0,1171},
  {"freeform",8,0,1,0,1173}
}
```

12.1.4.46 static KeyWord kw_42 () [static]

Initial value:

```cpp
{
  {"emulator_samples",9,0,1,1167},
  {"export_points_file",11,2,3,0,1175,kw_44},
  {"import_points_file",11,2,2,0,1169,kw_41},
  {"metropolis",8,2,5,0,1187,kw_37,0,0,0,0,0,"{Metropolis type for
    the MCMC algorithm} MethodCommands.html#MethodNonDBayesCalib"},
  {"proposal_covariance_scale",14,0,7,0,1199,0,0,0,0,0,"{Proposal
    covariance scaling} MethodCommands.html#MethodNonDBayesCalib"},
  {"rejection",8,2,4,0,1181,kw_38},
  {"rng",8,2,6,0,1193,kw_39,0,0,0,0,0,"{Random seed generator} Met
    hodCommands.html#MethodNonDBayesCalib"}
}
```

12.1.4.47 static KeyWord kw_43 () [static]

Initial value:

```cpp
{
  {"annotated",8,0,1,0,1153},
  {"freeform",8,0,1,0,1155}
}
```

12.1.4.48 static KeyWord kw_44 () [static]

Initial value:

```cpp
{
  {"annotated",8,0,1,0,1147},
  {"freeform",8,0,1,0,1149}
}
```

12.1.4.49 static KeyWord kw_45 () [static]

Initial value:

```cpp
{
  {"dakota",8,0,1,1,1141},
  {"annotated",8,0,1,0,1141},
  {"freeform",8,0,1,0,1141}
}
```
12.1.4.50 static KeyWord kw_46 () [static]

Initial value:

{
    {"sparse_grid_level",13,0,1,0,1159}
}

12.1.4.51 static KeyWord kw_47 () [static]

Initial value:

{
    {"sparse_grid_level",13,0,1,0,1163}
}

12.1.4.52 static KeyWord kw_48 () [static]

Initial value:

{
    {"gaussian_process",8,5,1,1,1137,kw_45},
    {"kriging",0,5,1,1,1136,kw_45},
    {"pce",8,1,1,1,1157,kw_46},
    {"sc",8,1,1,1,1161,kw_47}
}

12.1.4.53 static KeyWord kw_49 () [static]

Initial value:

{
    {"emulator",8,4,1,0,1135,kw_48},
    {"metropolis",8,2,3,0,1187,kw_37,0.,0.,0.,0.,0.,0.,0.,0.,0.},{Metropolis type for the MCMC algorithm} MethodCommands.html#MethodNonDBayesCalib"},
    {"proposal_covariance_scale",14,0,5,0,1199,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.},{Proposal covariance scaling} MethodCommands.html#MethodNonDBayesCalib"},
    {"rejection",8,2,2,0,1181,kw_38},
    {"rng",8,2,4,0,1193,kw_39,0.,0.,0.,0.,0.,0.},{Random seed generator} MethodCommands.html#MethodNonDBayesCalib"}
}
12.1.4.54 static KeyWord kw_50 () [static]

Initial value:

```
{ "calibrate_sigma", 8, 0, 4, 0, 1247, 0, 0, 0, 0, 0, 0, "[Calibrate sigma flag] MethodCommands.html#MethodNonDBayesCalib" },
{ "dream", 8, 6, 1, 1, 1201, kw_36 },
{ "gpm3a", 8, 7, 1, 1, 1165, kw_42 },
{ "likelihood_scale", 10, 0, 3, 0, 1245, 0, 0, 0, 0, 0, 0, "[Likelihood scale factor] MethodCommands.html#MethodNonDBayesCalib" },
{ "queso", 8, 5, 1, 1, 1133, kw_49 },
{ "samples", 9, 0, 6, 0, 1303, 0, 0, 0, 0, 0, 0, "[Number of samples] MethodCommands.html#MethodNonDMC" },
{ "seed", 0x19, 0, 5, 0, 1305, 0, 0, 0, 0, 0, 0, "[Random seed] MethodCommands.html#MethodEG" },
{ "use_derivatives", 8, 0, 2, 0, 1243 }
```

12.1.4.55 static KeyWord kw_51 () [static]

Initial value:

```
{ "deltas_per_variable", 5, 0, 2, 2, 1526 },
{ "step_vector", 14, 0, 1, 1, 1525, 0, 0, 0, 0, 0, 0, "[Step vector] MethodCommands.html#MethodPSCPS" },
{ "steps_per_variable", 13, 0, 2, 2, 1527, 0, 0, 0, 0, 0, 0, "[Number of steps per variable] MethodCommands.html#MethodPSCPS" }
```

12.1.4.56 static KeyWord kw_52 () [static]

Initial value:

```
{ "beta_solver_name", 11, 0, 1, 1, 569 },
{ "misc_options", 15, 0, 5, 0, 577, 0, 0, 0, 0, 0, 0, "[Specify miscellaneous options] MethodCommands.html#MethodSCOLIBDC" },
{ "seed", 0x19, 0, 3, 0, 573, 0, 0, 0, 0, 0, 0, "[Random seed for stochastic pattern search] MethodCommands.html#MethodSCOLIBPS" },
{ "show_misc_options", 8, 0, 4, 0, 575, 0, 0, 0, 0, 0, 0, "[Show miscellaneous options] MethodCommands.html#MethodSCOLIBDC" },
{ "solution_accuracy", 2, 0, 2, 0, 570 },
{ "solution_target", 10, 0, 2, 0, 571, 0, 0, 0, 0, 0, 0, "[Desired solution target] MethodCommands.html#MethodSCOLIBDC" }
```

12.1.4.57 static KeyWord kw_53 () [static]

Initial value:
12.1 Dakota Namespace Reference

{ "initial_delta", 10, 0, 5, 0, 487, 0, 0, 0, 0, 0, 0, "{Initial offset value
MethodCommands.html#MethodSCOLIBPS"},
"misc_options", 15, 0, 4, 0, 577, 0, 0, 0, 0, 0, 0, "{Specify miscellaneous
options} MethodCommands.html#MethodSCOLIBDC"},
{"seed", 0x19, 0, 2, 0, 573, 0, 0, 0, 0, 0, 0, "{Random seed for stochastic
pattern search} MethodCommands.html#MethodSCOLIBPS"},
{"show_misc_options", 8, 0, 3, 0, 575, 0, 0, 0, 0, 0, 0, "{Show miscellaneous
options} MethodCommands.html#MethodSCOLIBDC"},
{"solution_accuracy", 2, 0, 1, 0, 570},
{"solution_target", 10, 0, 1, 0, 571, 0, 0, 0, 0, 0, 0, "{Desired solution target
MethodCommands.html#MethodSCOLIBDC"},
{"threshold_delta", 10, 0, 6, 0, 489, 0, 0, 0, 0, 0, 0, "{Threshold for offset
values} MethodCommands.html#MethodSCOLIBPS"}
}

12.1.4.58 static KeyWord kw_54 () [static]

Initial value:

{ "all_dimensions", 8, 0, 1, 1, 497},
{ "major_dimension", 8, 0, 1, 1, 495 }

12.1.4.59 static KeyWord kw_55 [static]

Initial value:

{ "constraint_penalty", 10, 0, 6, 0, 507, 0, 0, 0, 0, 0, 0, "{Constraint penalty
MethodCommands.html#MethodSCOLIBDIR"},
{ "division", 8, 2, 1, 0, 493, kw_54, 0, 0, 0, 0, 0, "{Box subdivision approach
MethodCommands.html#MethodSCOLIBDIR"},
{ "global_balance_parameter", 10, 0, 2, 0, 499, 0, 0, 0, 0, 0, 0, "{Global search balancing parameter
MethodCommands.html#MethodSCOLIBDIR"},
{ "local_balance_parameter", 10, 0, 3, 0, 501, 0, 0, 0, 0, 0, 0, "{Local search balancing parameter
MethodCommands.html#MethodSCOLIBDIR"},
{ "max_boxsize_limit", 10, 0, 4, 0, 503, 0, 0, 0, 0, 0, 0, "{Maximum boxsize limit
MethodCommands.html#MethodSCOLIBDIR"},
{ "min_boxsize_limit", 10, 0, 5, 0, 505, 0, 0, 0, 0, 0, 0, "{Minimum boxsize limit
MethodCommands.html#MethodSCOLIBDIR"},
{ "misc_options", 15, 0, 10, 0, 577, 0, 0, 0, 0, 0, 0, "{Specify miscellaneous
options} MethodCommands.html#MethodSCOLIBDC"},
{ "seed", 0x19, 0, 8, 0, 573, 0, 0, 0, 0, 0, 0, "{Random seed for stochastic
pattern search} MethodCommands.html#MethodSCOLIBPS"},
{ "show_misc_options", 8, 0, 9, 0, 575, 0, 0, 0, 0, 0, 0, "{Show miscellaneous
options} MethodCommands.html#MethodSCOLIBDC"},
{ "solution_accuracy", 2, 0, 7, 0, 570},
{ "solution_target", 10, 0, 7, 0, 571, 0, 0, 0, 0, 0, 0, "{Desired solution target
MethodCommands.html#MethodSCOLIBDC"}
}

12.1.4.60 static KeyWord kw_56 () [static]

Initial value:
12.1.4.61 static KeyWord kw_57 () [static]
Initial value:

```
{ "linear_rank", 8, 0, 1, 1, 523 },
{ "merit_function", 8, 0, 1, 1, 525 }
```

12.1.4.62 static KeyWord kw_58 () [static]
Initial value:

```
{ "flat_file", 11, 0, 1, 1, 519 },
{ "simple_random", 8, 0, 1, 1, 515 },
{ "unique_random", 8, 0, 1, 1, 517 }
```

12.1.4.63 static KeyWord kw_59 () [static]
Initial value:

```
{ "mutation_range", 9, 0, 2, 0, 561, 0, 0, 0, 0, 0, 0, "{Mutation range} MethodCommands.html#MethodSCOLIBEA" },
{ "mutation_scale", 10, 0, 1, 0, 559, 0, 0, 0, 0, 0, 0, "{Mutation scale} MethodCommands.html#MethodSCOLIBEA" }
```

12.1.4.64 static KeyWord kw_60 () [static]
Initial value:

```
{ "non_adaptive", 8, 0, 2, 0, 563, 0, 0, 0, 0, 0, 0, "{Non-adaptive mutation flag} MethodCommands.html#MethodSCOLIBEA" },
{ "offset_cauchy", 8, 2, 1, 1, 555, kw_59 },
{ "offset_normal", 8, 2, 1, 1, 553, kw_59 },
{ "offset_uniform", 8, 2, 1, 1, 557, kw_59 },
{ "replace_uniform", 8, 0, 1, 1, 551 }
```
12.1.4.65 static KeyWord kw_61 () [static]

Initial value:

```cpp
{
    "chc",9,0,1,1,531,0,0.,0.,0.,0.,0., "{CHC replacement type} MethodCommands.html#MethodSCOLIBEA",
    "elitist",9,0,1,1,533,0,0.,0.,0.,0.,0., "{Elitist replacement type} MethodCommands.html#MethodSCOLIBEA",
    "new_solutions_generated",9,0,2,0,535,0,0.,0.,0.,0.,0., "{New solutions generated} MethodCommands.html#MethodSCOLIBEA",
    "random",9,0,1,1,529,0,0.,0.,0.,0.,0., "{Random replacement type} MethodCommands.html#MethodSCOLIBEA"
}
```

12.1.4.66 static KeyWord kw_62 () [static]

Initial value:

```cpp
{
    "constraint_penalty",10,0,9,0,565,
    "crossover_rate",10,0,5,0,537,0,0.,0.,0.,0.,0., "{Crossover rate} MethodCommands.html#MethodSCOLIBEA",
    "crossover_type",8,5,6,0,539,kw_56,0,0.,0.,0.,0.,0., "{Crossover type} MethodCommands.html#MethodSCOLIBEA",
    "fitness_type",8,2,3,0,521,kw_57,0,0.,0.,0.,0.,0., "{Fitness type} MethodCommands.html#MethodSCOLIBEA",
    "initialization_type",8,3,2,0,513,kw_58,0,0.,0.,0.,0.,0., "{Initialization type} MethodCommands.html#MethodSCOLIBEA",
    "misc_options",15,0,13,0,577,0,0.,0.,0.,0.,0., "{Specify miscellaneous options} MethodCommands.html#MethodSCOLIBDC",
    "mutation_rate",10,0,7,0,547,0,0.,0.,0.,0.,0., "{Mutation rate} MethodCommands.html#MethodSCOLIBEA",
    "mutation_type",8,5,8,0,549,kw_60,0,0.,0.,0.,0., "{Mutation type} MethodCommands.html#MethodSCOLIBEA",
    "population_size",0x19,0,1,0,511,0,0.,0.,0.,0.,0., "{Number of population members} MethodCommands.html#MethodSCOLIBEA",
    "replacement_type",8,4,4,0,527,kw_61,0,0.,0.,0.,0.,0., "{Replacement type} MethodCommands.html#MethodSCOLIBEA",
    "seed",0x19,0,11,0,573,0,0.,0.,0.,0.,0., "{Random seed for stochastic pattern search} MethodCommands.html#MethodSCOLIBPS",
    "solution_accuracy",2,0,10,0,570,
    "solution_target",10,0,10,0,571,0,0.,0.,0.,0.,0., "{Desired solution target} MethodCommands.html#MethodSCOLIBDC"
}
```

12.1.4.67 static KeyWord kw_63 () [static]

Initial value:

```cpp
{
    "adaptive_pattern",8,0,1,1,461,
    "basic_pattern",8,0,1,1,463,
    "multi_step",8,0,1,1,459
}
```
12.1.4.68 static KeyWord kw_64 () [static]

Initial value:

```c
{
    \{"coordinate",8,0,1,1,449},
    \{"simplex",8,0,1,1,451}
}
```

12.1.4.69 static KeyWord kw_65 () [static]

Initial value:

```c
{
    \{"blocking",8,0,1,1,467},
    \{"nonblocking",8,0,1,1,469}
}
```

12.1.4.70 static KeyWord kw_66 () [static]

Initial value:

```c
{
    \{"constant_penalty",8,0,1,0,441,0,0,0,0,0,0,"\{Control of dynamic penalty\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"constraint_penalty",10,0,16,0,483,0,0,0,0,0,0,"\{Constraint penalty\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"contraction_factor",10,0,15,0,481,0,0,0,0,0,0,"\{Pattern contraction factor\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"expand_after_success",9,0,3,0,445,0,0,0,0,0,0,"\{Number of consecutive improvements before expansion\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"exploratory_moves",8,3,7,0,457,kw_63,0,0,0,0,0,0,"\{Exploratory moves selection\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"initial_delta",10,0,13,0,487,0,0,0,0,0,0,"\{Initial offset value\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"misc_options",15,0,12,0,577,0,0,0,0,0,0,"\{Specify miscellaneous options\}\ MethodCommands.html\#MethodSCOLIBDC"},
    \{"no_expansion",8,0,2,0,443,0,0,0,0,0,0,"\{No expansion flag\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"pattern_basis",8,2,4,0,447,kw_64,0,0,0,0,0,0,"\{Pattern basis selection\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"seed",0x19,0,10,0,573,0,0,0,0,0,0,"\{Random seed for stochastic pattern search\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"show_misc_options",8,0,11,0,575,0,0,0,0,0,0,"\{Show miscellaneous options\}\ MethodCommands.html\#MethodSCOLIBDC"},
    \{"solution_accuracy",2,0,9,0,570},
    \{"solution_target",10,0,9,0,571,0,0,0,0,0,0,"\{Desired solution target\}\ MethodCommands.html\#MethodSCOLIBDC"},
    \{"stochastic",8,0,5,0,453,0,0,0,0,0,0,"\{Stochastic pattern search\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"synchronization",8,2,8,0,465,kw_65,0,0,0,0,0,0,"\{Evaluation synchronization\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"threshold_delta",10,0,14,0,489,0,0,0,0,0,0,"\{Threshold for offset set values\}\ MethodCommands.html\#MethodSCOLIBPS"},
    \{"total_pattern_size",9,0,6,0,455,0,0,0,0,0,0,"\{Total number of points in pattern\}\ MethodCommands.html\#MethodSCOLIBPS"}
}
```
12.1.4.71 static KeyWord kw_67 () [static]

Initial value:

```c
{
  "constant_penalty", 8, 0, 4, 0, 479, 0, 0, 0, 0, 0, 0, 0,"{Control of dynamic
  penalty} MethodCommands.html#MethodSCOLIBSW",
  "constraint_penalty", 10, 0, 12, 0, 483, 0, 0, 0, 0, 0, 0,"{Constraint pen-
  alty} MethodCommands.html#MethodSCOLIBSW",
  "contract_after_failure", 9, 0, 1, 0, 473, 0, 0, 0, 0, 0, 0,"{Number of con-
 secutive failures before contraction} MethodCommands.html#MethodSCOLIBSW",
  "contraction_factor", 10, 0, 11, 0, 481, 0, 0, 0, 0, 0, 0,"{Pattern contra-
  ction factor} MethodCommands.html#MethodSCOLIBPS",
  "expand_after_success", 9, 0, 3, 0, 477, 0, 0, 0, 0, 0, 0,"{Number of cons-
  ecutive improvements before expansion} MethodCommands.html#MethodSCOLIBSW",
  "initial_delta", 10, 0, 9, 0, 487, 0, 0, 0, 0, 0, 0,"{Initial offset value} Met-
  hodCommands.html#MethodSCOLIBPS",
  "misc_options", 15, 0, 8, 0, 577, 0, 0, 0, 0, 0, 0,"{Specify miscellaneous
  options} MethodCommands.html#MethodSCOLIBDC",
  "no_expansion", 8, 0, 2, 0, 475, 0, 0, 0, 0, 0, 0,"{No expansion flag} Met-
  hodCommands.html#MethodSCOLIBSW",
  "seed", 0x19, 0, 6, 0, 673, 0, 0, 0, 0, 0, 0,"{Random seed for stochastic
  pattern search} MethodCommands.html#MethodSCOLIBPS",
  "show_misc_options", 8, 0, 7, 0, 575, 0, 0, 0, 0, 0, 0,"{Show miscellaneous
  options} MethodCommands.html#MethodSCOLIBDC",
  "solution_accuracy", 2, 0, 5, 0, 570, 0, 0, 0, 0, 0, 0,"{Desired solution t-
  arget} MethodCommands.html#MethodSCOLIBDC",
  "threshold_delta", 10, 0, 10, 0, 489, 0, 0, 0, 0, 0, 0,"{Threshold for off-
  set values} MethodCommands.html#MethodSCOLIBPS"
}
```

12.1.4.72 static KeyWord kw_68 () [static]

Initial value:

```c
{
  "frcg", 8, 0, 1, 1, 1, 185,
  "linear_inequality_constraint_matrix", 14, 0, 7, 0, 431, 0, 0, 0, 0, 0, 0,"{Linear
  inequality coefficient matrix} MethodCommands.html#MethodIndControl",
  "linear_inequality_scale_types", 15, 0, 9, 0, 435, 0, 0, 0, 0, 0, 0,"{Linear
  inequality scaling types} MethodCommands.html#MethodIndControl",
  "linear_inequality_scales", 14, 0, 10, 0, 437, 0, 0, 0, 0, 0, 0,"{Linear inequ
  ality scales} MethodCommands.html#MethodIndControl",
  "linear_inequality_targets", 14, 0, 8, 0, 433, 0, 0, 0, 0, 0, 0,"{Linear inequ
  ality targets} MethodCommands.html#MethodIndControl",
  "linear_inequality_constraint_matrix", 14, 0, 2, 0, 421, 0, 0, 0, 0, 0, 0,"{Linear
  inequality coefficient matrix} MethodCommands.html#MethodIndControl",
  "linear_inequality_lower_bounds", 14, 0, 3, 0, 423, 0, 0, 0, 0, 0, 0,"{Linear
  inequality lower bounds} MethodCommands.html#MethodIndControl",
  "linear_inequality_scale_types", 15, 0, 5, 0, 427, 0, 0, 0, 0, 0, 0,"{Linear
  inequality scaling types} MethodCommands.html#MethodIndControl",
  "linear_inequality_scales", 14, 0, 6, 0, 429, 0, 0, 0, 0, 0, 0,"{Linear inequ
  ality scales} MethodCommands.html#MethodIndControl",
  "linear_inequality_upper_bounds", 14, 0, 4, 0, 425, 0, 0, 0, 0, 0, 0,"{Linear inequ
  ality upper bounds} MethodCommands.html#MethodIndControl",
  "mfd", 8, 0, 1, 1, 187
}
```
12.1.4.73 static KeyWord kw_69 () [static]

Initial value:

```
{
  "linear_equality_constraint_matrix",14,0,7,0,431,0,0.,0.,0.,0.,0,"{Linear equality coefficient matrix} MethodCommands.html#MethodIndControl"},
  "linear_equality_scale_types",15,0,9,0,435,0,0.,0.,0.,0.,0,"{Linear equality scaling types} MethodCommands.html#MethodIndControl"},
  "linear_equality_scales",14,0,10,0,437,0,0.,0.,0.,0.,0,"{Linear equality scales} MethodCommands.html#MethodIndControl"},
  "linear_equality_targets",14,0,8,0,433,0,0.,0.,0.,0,"{Linear equality targets} MethodCommands.html#MethodIndControl"},
  "linear_inequality_constraint_matrix",14,0,2,0,421,0,0.,0.,0.,0,0,"{Linear inequality coefficient matrix} MethodCommands.html#MethodIndControl"],
  "linear_inequality_lower_bounds",14,0,3,0,423,0,0.,0.,0.,0,0,"{Linear inequality lower bounds} MethodCommands.html#MethodIndControl"],
  "linear_inequality_scale_types",15,0,5,0,427,0,0.,0.,0.,0,0,"{Linear inequality scaling types} MethodCommands.html#MethodIndControl"],
  "linear_inequality_scales",14,0,6,0,429,0,0.,0.,0.,0,"{Linear inequality scales} MethodCommands.html#MethodIndControl"],
  "linear_inequality_upper_bounds",14,0,4,0,425,0,0.,0.,0.,0,"{Linear inequality upper bounds} MethodCommands.html#MethodIndControl"}
```

12.1.4.74 static KeyWord kw_70 () [static]

Initial value:

```
{
  "drop_tolerance",10,0,1,0,1271}
}
```

12.1.4.75 static KeyWord kw_71 () [static]

Initial value:

```
{
  "box_benchiken",8,0,1,1,1261,0,0.,0.,0.,0,0,"{CHOOSE DACE type}"},
  "central_composite",8,0,1,1,1263},
  "fixed_seed",8,0,5,0,1273,0,0.,0.,0.,0,"{Fixed seed flag} MethodCommands.html#MethodDDACE"},
  "grid",8,0,1,1,1251},
  "lhs",8,0,1,1,1257},
  "main_effects",8,0,2,0,1265,0,0.,0.,0.,0,"{Main effects} MethodCommands.html#MethodDDACE"},
  "oa_lhs",8,0,1,1,1259},
  "oas",8,0,1,1,1255},
  "quality_metrics",8,0,3,0,1267,0,0.,0.,0.,0,"{Quality metrics} MethodCommands.html#MethodDDACE"},
  "random",8,0,1,1,1253},
  "samples",9,0,8,0,1303,0,0.,0.,0.,0,"{Number of samples} MethodCommands.html#MethodDDACE"},
  "seed",0x19,0,7,0,1305,0,0.,0.,0.,0,"{Random seed} MethodCommands.html#MethodDDACE"},
  "symbols",9,0,6,0,1275,0,0.,0.,0.,0,"{Number of symbols} MethodCommands.html#MethodDDACE"}
```
12.1 Dakota Namespace Reference

12.1.4.76 static KeyWord kw_72 () [static]

Initial value:

```
{
    "bfgs", 8, 0, 1, 1, 173,
    "frcg", 8, 0, 1, 1, 169,
    "linear_equality_constraint_matrix", 14, 0, 7, 0, 431, 0, 0, 0, 0, 0, 0, 0, 0,
    "linear_equality_scale_types", 15, 0, 9, 0, 435, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    "linear_equality_scales", 14, 0, 10, 0, 437, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    "linear_equality_targets", 14, 0, 8, 0, 433, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    "linear_inequality_constraint_matrix", 14, 0, 2, 0, 421, 0, 0, 0, 0, 0, 0, 0, 0,
    "linear_inequality_scale_types", 15, 0, 5, 0, 427, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    "linear_inequality_scales", 14, 0, 6, 0, 429, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    "linear_inequality_targets", 14, 0, 4, 0, 425, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    "mmfd", 8, 0, 1, 1, 171,
    "slp", 8, 0, 1, 1, 175,
    "sqp", 8, 0, 1, 1, 177
}
```

12.1.4.77 static KeyWord kw_73 () [static]

Initial value:

```
{
    "annotated", 8, 0, 1, 0, 629,
    "freeform", 8, 0, 1, 0, 631
}
```

12.1.4.78 static KeyWord kw_74 () [static]

Initial value:

```
{
    "dakota", 8, 0, 1, 1, 617,
    "surfpack", 8, 0, 1, 1, 615
}
```
12.1.4.79 static KeyWord kw_75 () [static]

Initial value:

```
{
    {"annotated", 8, 0, 1, 0, 623},
    {"freeform", 8, 0, 1, 0, 625}
}
```

12.1.4.80 static KeyWord kw_76 () [static]

Initial value:

```
{
    {"export_points_file", 11, 2, 4, 0, 627, kw_73, 0, 0, 0, 0, "(File name for exporting approximation-based samples from evaluating the GP) MethodCommands.html#MethodEG"},
    {"gaussian_process", 8, 2, 1, 0, 613, kw_74, 0, 0, 0, 0, "(GP selection) MethodCommands.html#MethodEG"},
    {"import_points_file", 11, 2, 3, 0, 621, kw_75, 0, 0, 0, 0, "(File name for points to be imported as the basis for the initial GP) MethodCommands.html#MethodEG"},
    {"kriging", 0, 2, 1, 0, 612, kw_74},
    {"seed", 0x19, 0, 5, 0, 1305, 0, 0, 0, 0, "(Random seed) MethodCommands.html#MethodEG"},
    {"use_derivatives", 8, 0, 2, 0, 619, 0, 0, 0, 0, 0, "(Derivative usage) MethodCommands.html#MethodEG"}
}
```

12.1.4.81 static KeyWord kw_77 () [static]

Initial value:

```
{
    {"batch_size", 9, 0, 2, 0, 1027},
    {"distribution", 8, 2, 5, 0, 1075, kw_15, 0, 0, 0, 0, "(Distribution type) MethodCommands.html#MethodNonD"},
    {"emulator_samples", 9, 0, 1, 0, 1025},
    {"gen_reliability_levels", 14, 1, 7, 0, 1085, kw_16, 0, 0, 0, 0, "(Generalized reliability levels) MethodCommands.html#MethodNonD"},
    {"probability_levels", 14, 1, 6, 0, 1081, kw_17, 0, 0, 0, 0, "(Probability levels) MethodCommands.html#MethodNonD"},
    {"rng", 8, 2, 8, 0, 1089, kw_18, 0, 0, 0, 0, "(Random number generator) MethodCommands.html#MethodNonDMC"},
    {"samples", 9, 0, 4, 0, 1303, 0, 0, 0, 0, 0, "(Number of samples) MethodCommands.html#MethodNonDMC"},
    {"seed", 0x19, 0, 3, 0, 1305, 0, 0, 0, 0, 0, "(Random seed) MethodCommands.html#MethodEG"
}
```

12.1.4.82 static KeyWord kw_78 () [static]

Initial value:
12.1.4.83 static KeyWord kw_79 () [static]

Initial value:

```
{ "drop_tolerance", 10, 0, 1, 1285 }
```

12.1.4.84 static KeyWord kw_80 () [static]

Initial value:

```
{ "fixed_seed", 8, 0, 4, 1287, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Fixed seed flag" MethodCommands.html#MethodFSUDACE"},
{ "latinize", 8, 0, 1, 1279, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Latinization of samples" MethodCommands.html#MethodFSUDACE"},
{ "num_trials", 9, 0, 6, 1297, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Number of trials" MethodCommands.html#MethodFSUDACE"},
{ "quality_metrics", 8, 0, 2, 1281, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Quality metrics" MethodCommands.html#MethodFSUDACE"},
{ "samples", 9, 0, 8, 1303, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Number of samples" MethodCommands.html#MethodNonDMC"},
{ "seed", 0x19, 0, 7, 1305, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Random seed" MethodCommands.html#MethodNonDMC"},
{ "trial_type", 8, 3, 5, 1289, kw_78, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Trial type" MethodCommands.html#MethodFSUDACE"},
{ "variance_based_decomp", 8, 1, 3, 1283, kw_79, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Variance based decomposition" MethodCommands.html#MethodFSUDACE"} }
```

12.1.4.85 static KeyWord kw_81 () [static]

Initial value:

```
{ "drop_tolerance", 10, 0, 1, 0, 1493 }
```

12.1.4.86 static KeyWord kw_82 () [static]

Initial value:

```
{ "fixed_sequence", 8, 0, 6, 0, 1497, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "Fixed sequence flag" MethodCommands.html#MethodFSUDACE"} }
```
12.1.4.87 static KeyWord kw_83 () [static]

Initial value:

```c
{
  "annotated", 8, 0, 1, 0, 931},
  "freeform", 8, 0, 1, 0, 933}
```

12.1.4.88 static KeyWord kw_84 () [static]

Initial value:

```c
{
  "annotated", 8, 0, 1, 0, 925},
  "freeform", 8, 0, 1, 0, 927}
```

12.1.4.89 static KeyWord kw_85 () [static]

Initial value:

```c
{
  "parallel", 8, 0, 1, 1, 949},
  "series", 8, 0, 1, 1, 947}
```

12.1.4.90 static KeyWord kw_86 () [static]

Initial value:
12.1 Dakota Namespace Reference

{ "gen_reliabilities",8,0,1,1,943},
{ "probabilities",8,0,1,1,941},
{ "system",8,2,2,0,945,kw_85 }
}

12.1.4.91 static KeyWord kw_87 () [static]
Initial value:

{ "compute",8,3,2,0,939,kw_86},
{ "num_response_levels",13,0,1,0,937 }
}

12.1.4.92 static KeyWord kw_88 () [static]
Initial value:

{ "distribution",8,2,7,0,1075,kw_15,0.,0.,0.,0,"{Distribution type
} MethodCommands.html#MethodNonD"},
{ "emulator_samples",9,0,1,0,921},
{ "export_points_file",11,2,3,0,929,kw_83,0.,0.,0.,0,"{File name f
or exporting approximation-based samples from evaluating the emulator} MethodComm
ands.html#MethodNonDBayesCalib"},
{ "gen_reliability_levels",14,1,9,0,1085,kw_16,0.,0.,0.,0,"{Genera
lized reliability levels} MethodCommands.html#MethodNonD"},
{ "import_points_file",11,2,2,0,923,kw_84,0.,0.,0.,0,"{File contai
ning points to evaluate} MethodCommands.html#MethodPSLPS"},
{ "probability_levels",14,1,6,0,1081,kw_17,0.,0.,0.,0,"{Probabilit
y levels} MethodCommands.html#MethodNonD"},
{ "response_levels",14,2,4,0,935,kw_87},
{ "rng",8,2,10,0,1089,kw_18,0.,0.,0.,0,"{Random number generator} MethodCommands.html#MethodNonDMC"},
{ "samples",9,0,6,0,1303,0.,0.,0.,0,"{Number of samples} MethodC
ommands.html#MethodNonDMC"},
{ "seed",0x19,0,5,0,1305,0.,0.,0.,0,"{Random seed} MethodCommands.h
}

12.1.4.93 static KeyWord kw_89 () [static]
Initial value:

{ "parallel",8,0,1,1,1073},
{ "series",8,0,1,1,1071 }
}
12.1.4.94 static KeyWord kw_90 () [static]

Initial value:

{
   {"gen_reliabilities",8,0,1,1,1067},
   {"probabilities",8,0,1,1,1065},
   {"system",8,2,2,0,1069,kw_89}
}

12.1.4.95 static KeyWord kw_91 () [static]

Initial value:

{
   {"compute",8,3,2,0,1063,kw_90},
   {"num_response_levels",13,0,1,0,1061}
}

12.1.4.96 static KeyWord kw_92 () [static]

Initial value:

{
   {"annotated",8,0,1,0,1051},
   {"freeform",8,0,1,0,1053}
}

12.1.4.97 static KeyWord kw_93 () [static]

Initial value:

{
   {"dakota",8,0,1,1,1039},
   {"surfpack",8,0,1,1,1037}
}

12.1.4.98 static KeyWord kw_94 () [static]

Initial value:

{
   {"annotated",8,0,1,0,1045},
   {"freeform",8,0,1,0,1047}
}
12.1 Dakota Namespace Reference 169

12.1.4.99 static KeyWord kw_95 () [static]

Initial value:

```
{"export_points_file",11,2,4,0,1049,kw_92},
{"gaussian_process",8,2,1,0,1035,kw_93},
{"import_points_file",11,2,3,0,1043,kw_94},
{"kriging",0,2,1,0,1034,kw_93},
{"use_derivatives",8,0,2,0,1041}
```

12.1.4.100 static KeyWord kw_96 () [static]

Initial value:

```
{"distribution",8,2,5,0,1075,kw_15,0.,0.,0.,0.,"{Distribution type
} MethodCommands.html#MethodNonD"},
{"ea",8,0,1,0,1055},
{"ego",8,5,1,0,1013,kw_95},
{"gen_reliability_levels",14,1,7,0,1085,kw_16,0.,0.,0.,0.,"{Generalized reliability levels} MethodCommands.html#MethodNonD"},
{"lhs",8,0,1,0,1057},
{"probability_levels",14,1,6,0,1081,kw_17,0.,0.,0.,0.,"{Probability levels} MethodCommands.html#MethodNonD"},
{"response_levels",14,2,2,0,1059,kw_91},
{"rng",8,2,8,0,1089,kw_18,0.,0.,0.,0.,"{Random number generator} MethodCommands.html#MethodNonDMC"},
{"samples",9,0,4,0,1303,0.,0.,0.,0.,"{Number of samples} MethodCommands.html#MethodNonDMC"},
{"sbo",8,5,1,0,1031,kw_95},
{"seed",0x19,0,3,0,1305,0.,0.,0.,0.,0.,"{Random seed} MethodCommands.html#MethodEG"}
```

12.1.4.101 static KeyWord kw_97 () [static]

Initial value:

```
{"mt19937",8,0,1,1,1127},
{"rnum2",8,0,1,1,1129}
```

12.1.4.102 static KeyWord kw_98 () [static]

Initial value:

```
{"annotated",8,0,1,0,1117},
{"freeform",8,0,1,0,1119}
```
12.1.4.103  static KeyWord kw_99 () [static]

Initial value:

```
{
    {"dakota", 8, 0, 1, 1, 1105},
    {"surfpack", 8, 0, 1, 1, 1103}
}
```

12.1.4.104  static KeyWord kw_100 () [static]

Initial value:

```
{
    {"annotated", 8, 0, 1, 0, 1111},
    {"freeform", 8, 0, 1, 0, 1113}
}
```

12.1.4.105  static KeyWord kw_101 () [static]

Initial value:

```
{
    {"export_points_file", 11, 2, 4, 0, 1115, kw_98, 0., 0., 0., 0., 0., "{File name for exporting approximation-based samples from evaluating the GP} MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"gaussian_process", 8, 2, 1, 0, 1101, kw_99, 0., 0., 0., 0., "{EGO GP selection} MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"import_points_file", 11, 2, 3, 0, 1109, kw_100, 0., 0., 0., 0., 0., "{File name for points to be imported as the basis for the initial GP} MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"kriging", 0, 2, 1, 0, 1100, kw_99},
    {"use_derivatives", 8, 0, 2, 0, 1107, 0., 0., 0., 0., 0., 0., "{Derivative usage} MethodCommands.html#MethodNonDGlobalIntervalEst"},
}
```

12.1.4.106  static KeyWord kw_102 () [static]

Initial value:

```
{
    {"ea", 8, 0, 1, 0, 1121},
    {"ego", 8, 5, 1, 0, 1099, kw_101},
    {"lhs", 8, 0, 1, 0, 1123},
    {"rng", 8, 2, 2, 0, 1125, kw_97, 0., 0., 0., 0., 0., "{Random seed generator} MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"samples", 9, 0, 4, 0, 1303, 0., 0., 0., 0., 0., 0., "{Number of samples} MethodCommands.html#MethodNonDGlobalIntervalEst"},
    {"sbo", 8, 5, 1, 0, 1097, kw_101},
    {"seed", 0x19, 0, 3, 0, 1305, 0., 0., 0., 0., 0., 0., 0., "{Random seed} MethodCommands.html#MethodNonDGlobalIntervalEst"}
}
```
12.1.4.107 static KeyWord kw_103 () [static]
Initial value:
{
    "complementary", 8, 0, 1, 1, 1471,
    "cumulative", 8, 0, 1, 1, 1469
}

12.1.4.108 static KeyWord kw_104 () [static]
Initial value:
{
    "num_gen_reliability_levels", 13, 0, 1, 0, 1479
}

12.1.4.109 static KeyWord kw_105 () [static]
Initial value:
{
    "num_probability_levels", 13, 0, 1, 0, 1475
}

12.1.4.110 static KeyWord kw_106 () [static]
Initial value:
{
    "annotated", 8, 0, 1, 0, 1437,
    "freeform", 8, 0, 1, 0, 1439
}

12.1.4.111 static KeyWord kw_107 () [static]
Initial value:
{
    "annotated", 8, 0, 1, 0, 1431,
    "freeform", 8, 0, 1, 0, 1433
}

12.1.4.112 static KeyWord kw_108 () [static]
Initial value:
{
    "parallel", 8, 0, 1, 1, 1465,
    "series", 8, 0, 1, 1, 1463
}
12.1.4.113  static KeyWord kw_109 ()  [static]

Initial value:

    {
        "gen_reliabilities", 8, 0, 1, 1, 1459),
        "probabilities", 8, 0, 1, 1, 1457),
        "system", 8, 2, 2, 0, 1461, kw_108)
    }

12.1.4.114  static KeyWord kw_110 ()  [static]

Initial value:

    {
        "compute", 8, 3, 2, 0, 1455, kw_109),
        "num_response_levels", 13, 0, 1, 0, 1453)
    }

12.1.4.115  static KeyWord kw_111 ()  [static]

Initial value:

    {
        "mt19937", 8, 0, 1, 1, 1457),
        "rnum2", 8, 0, 1, 1, 1459)
    }

12.1.4.116  static KeyWord kw_112 ()  [static]

Initial value:

    {
        "dakota", 8, 0, 1, 0, 1427),
        "surfpack", 8, 0, 1, 0, 1425)
    }

12.1.4.117  static KeyWord kw_113 ()  [static]

Initial value:

    {
        "distribution", 8, 2, 8, 0, 1467, kw_103),
        "export_points_file", 11, 2, 3, 0, 1435, kw_106, 0., 0., 0., "{File name
        for exporting approximation-based samples from evaluating the GP} MethodCommands.html#MethodNonDGlobalRel"},
        "gen_reliability_levels", 14, 1, 10, 0, 1477, kw_104),
        "import_points_file", 11, 2, 2, 0, 1429, kw_107, 0., 0., 0., "{File name
        for points to be imported as the basis for the initial GP} MethodCommands.html#MethodNonDGlobalRel"},
12.1.4.118 static KeyWord kw_114 () [static]

Initial value:

{
    "parallel", 8, 0, 1, 1, 917,
    "series", 8, 0, 1, 1, 915
}

12.1.4.119 static KeyWord kw_115 () [static]

Initial value:

{
    "gen_reliabilities", 8, 0, 1, 1, 911,
    "probabilities", 8, 0, 1, 1, 909,
    "system", 8, 2, 2, 0, 913, kw_114
}

12.1.4.120 static KeyWord kw_116 () [static]

Initial value:

{
    "compute", 8, 3, 2, 0, 907, kw_115,
    "num_response_levels", 13, 0, 1, 0, 905
}

12.1.4.121 static KeyWord kw_117 () [static]

Initial value:

{
    "adapt_import", 8, 0, 1, 0, 899,
    "distribution", 8, 2, 5, 0, 1075, kw_15, 0, 0, 0, 0, "(Distribution type
} MethodCommands.html#MethodNonDNonD),
    "gen_reliability_levels", 14, 1, 7, 0, 1085, kw_16, 0, 0, 0, 0, "(Generalized reliability levels) MethodCommands.html#MethodNonD"),
12.1.4.122  static KeyWord kw_118 ()  [static]

Initial value:

```c
{
   "annotated",0,1,0,1519},
   "freeform",0,1,0,1521
}
```

12.1.4.123  static KeyWord kw_119 ()  [static]

Initial value:

```c
{
   "import_points_file",11,2,1,1517,kw_118},
   "list_of_points",14,0,1,1515,0,0,,0,,0,"{List of points to evaluate} MethodCommands.html#MethodPSLPS"
}
```

12.1.4.124  static KeyWord kw_120 ()  [static]

Initial value:

```c
{
   "complementary",8,0,1,1349},
   "cumulative",8,0,1,1347}
}
```

12.1.4.125  static KeyWord kw_121 ()  [static]

Initial value:

```c
{
   "num_gen_reliability_levels",13,0,1,1343}
}
```
12.1 Dakota Namespace Reference

12.1.4.126 static KeyWord kw_122 () [static]

Initial value:

```
{
  "num_probability_levels":13,0,1,0,1339
}
```

12.1.4.127 static KeyWord kw_123 () [static]

Initial value:

```
{
  "parallel":8,0,1,1,1335,
  "series":8,0,1,1,1333
}
```

12.1.4.128 static KeyWord kw_124 () [static]

Initial value:

```
{
  "gen_reliabilities":8,0,1,1,1329,
  "probabilities":8,0,1,1,1327,
  "system":8,2,2,0,1331,kw_123
}
```

12.1.4.129 static KeyWord kw_125 () [static]

Initial value:

```
{
  "compute":8,3,2,0,1325,kw_124,
  "num_response_levels":13,0,1,0,1323
}
```

12.1.4.130 static KeyWord kw_126 () [static]

Initial value:

```
{
  "distribution":8,2,5,0,1345,kw_120,
  "gen_reliability_levels":14,1,4,0,1341,kw_121,
  "nip":8,0,1,0,1319,
  "probability_levels":14,1,3,0,1337,kw_122,
  "response_levels":14,2,2,0,1321,kw_125,
  "sqp":8,0,1,0,1317
}
```
12.1.4.131 static KeyWord kw_127 () [static]
Initial value:

```c
{ "nip", 8, 0, 1, 0, 1355 },
{ "sqp", 8, 0, 1, 0, 1353 }
```

12.1.4.132 static KeyWord kw_128 () [static]
Initial value:

```c
{ "adapt_import", 8, 0, 1, 1, 1389 },
{ "import", 8, 0, 1, 1, 1387 },
{ "mm_adapt_import", 8, 0, 1, 1, 1391 },
{ "samples", 9, 0, 2, 0, 1393, 0, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. },
{ "seed", 0x19, 0, 3, 0, 1395, 0, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. },
```

12.1.4.133 static KeyWord kw_129 () [static]
Initial value:

```c
{ "first_order", 8, 0, 1, 1, 1381 },
{ "sample_refinement", 8, 5, 2, 0, 1385, kw_128 },
{ "second_order", 8, 0, 1, 1, 1383 }
```

12.1.4.134 static KeyWord kw_130 () [static]
Initial value:

```c
{ "integration", 8, 3, 3, 0, 1379, kw_129, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0. },
{ "nip", 8, 0, 2, 0, 1377 },
{ "no_approx", 8, 0, 1, 1, 1373 },
{ "sqp", 8, 0, 2, 0, 1375 },
{ "u_taylor_mean", 8, 0, 1, 1, 1363 },
{ "u_taylor_mpp", 8, 0, 1, 1, 1367 },
{ "u_two_point", 8, 0, 1, 1, 1361 },
{ "x_taylor_mean", 8, 0, 1, 1, 1361 },
{ "x_taylor_mpp", 8, 0, 1, 1, 1365 },
{ "x_two_point", 8, 0, 1, 1, 1369 }
```
12.1.4.135 static KeyWord kw_131 () [static]
Initial value:

```c
{
   {"num_reliability_levels",13,0,1,0,1417}
}
```

12.1.4.136 static KeyWord kw_132 () [static]
Initial value:

```c
{
   {"parallel",8,0,1,1,1413},
   {"series",8,0,1,1,1411}
}
```

12.1.4.137 static KeyWord kw_133 () [static]
Initial value:

```c
{
   {"gen_reliabilities",8,0,1,1,1407},
   {"probabilities",8,0,1,1,1403},
   {"reliabilities",8,0,1,1,1405},
   {"system",8,2,2,0,1409,kw_132}
}
```

12.1.4.138 static KeyWord kw_134 () [static]
Initial value:

```c
{
   {"compute",8,4,2,0,1401,kw_133},
   {"num_response_levels",13,0,1,0,1399}
}
```

12.1.4.139 static KeyWord kw_135 () [static]
Initial value:

```c
{
   {"distribution",8,2,4,0,1467,kw_103},
   {"gen_reliability_levels",14,1,6,0,1477,kw_104},
   {"mpp_search",8,10,1,0,1359,kw_130,0.,0.,0.,"{MPP search type}MethodCommands.html#MethodNonDLocalRel"},
   {"probability_levels",14,1,5,0,1473,kw_105},
   {"reliability_levels",14,1,3,0,1415,kw_131},
   {"response_levels",14,2,2,0,1397,kw_134}
}
```
12.1.4.140 static KeyWord kw_136 () [static]

Initial value:

```c
{ "display_all_evaluations", 8, 0, 6, 0, 297, 
  "display_format", 11, 0, 4, 0, 293, 
  "function_precision", 10, 0, 1, 0, 287, 0, 0, 0, 0, 
  "history_file", 11, 0, 3, 0, 291, 
  "linear_equality_constraint_matrix", 14, 0, 12, 0, 431, 0, 0, 0, 0, 0, 
  "linear_equality_coefficient_matrix", MethodCommands.html#MethodLSNL2SOL, 
  "linear_equality_scale_types", 16, 0, 14, 0, 435, 0, 0, 0, 0, 0, 
  "linear_equality_scales", MethodCommands.html#MethodIndControl, 
  "linear_equality_targets", MethodCommands.html#MethodIndControl, 
  "linear_inequality_constraint_matrix", 14, 0, 7, 0, 421, 0, 0, 0, 0, 0, 
  "linear_inequality_coefficient_matrix", MethodCommands.html#MethodIndControl, 
  "linear_inequality_scale_types", 16, 0, 11, 0, 429, 0, 0, 0, 0, 0, 
  "linear_inequality_scales", MethodCommands.html#MethodIndControl, 
  "linear_inequality_upper_bounds", MethodCommands.html#MethodIndControl, 
  "linear_inequality_upper_bounds", MethodCommands.html#MethodIndControl, 
  "seed", 0x19, 0, 2, 0, 289, 0, 0, 0, 0, 0, 
  "variable_neighborhood_search", 10, 0, 5, 0, 295 }
```

12.1.4.141 static KeyWord kw_137 () [static]

Initial value:

```c
{ "num_offspring", 0x19, 0, 2, 0, 399, 0, 0, 0, 0, 0, 
  "num_parents", 0x19, 0, 1, 0, 397, 0, 0, 0, 0, 0, 
  "crossover_rate", 10, 0, 2, 0, 401, 0, 0, 0, 0, 0, 
  "multi_point_binary", 9, 0, 1, 1, 389, 0, 0, 0, 0, 0, 
  "multi_point_parameterized_binary", MethodCommands.html#MethodJEGADC, 
  "multi_point_real", 9, 0, 1, 1, 393, 0, 0, 0, 0, 0, 
  "variable_neighborhood_search", 10, 0, 5, 0, 295 }
```

12.1.4.142 static KeyWord kw_138 () [static]

Initial value:

```c
{ "crossover_rate", 10, 0, 2, 0, 401, 0, 0, 0, 0, 0, 
  "multi_point_binary", 9, 0, 1, 1, 389, 0, 0, 0, 0, 0, 
  "multi_point_parameterized_binary", MethodCommands.html#MethodJEGADC, 
  "multi_point_real", 9, 0, 1, 1, 393, 0, 0, 0, 0, 0, 
  "variable_neighborhood_search", 10, 0, 5, 0, 295 }
```
12.1.4.143 static KeyWord kw_139 () [static]
Initial value:

```
{
    "flat_file", 11, 0, 1, 1, 385,
    "simple_random", 8, 0, 1, 1, 381,
    "unique_random", 8, 0, 1, 1, 383
}
```

12.1.4.144 static KeyWord kw_140 () [static]
Initial value:

```
{
    "mutation_scale", 10, 0, 1, 0, 415, 0, 0., 0., 0., 0,
    "mutation_rate", 10, 0, 2, 0, 417, 0, 0., 0., 0., 0,
    "offset_cauchy", 8, 1, 1, 1, 411, kw_140,
    "offset_normal", 8, 1, 1, 1, 409, kw_140,
    "offset_uniform", 8, 1, 1, 1, 413, kw_140,
    "replace_uniform", 8, 0, 1, 1, 407
}
```

12.1.4.145 static KeyWord kw_141 () [static]
Initial value:

```
{
    "bit_random", 8, 0, 1, 1, 405,
    "mutation_rate", 10, 0, 2, 0, 417, 0, 0., 0., 0., 0,
    "offset_cauchy", 8, 1, 1, 1, 411, kw_140,
    "offset_normal", 8, 1, 1, 1, 409, kw_140,
    "offset_uniform", 8, 1, 1, 1, 413, kw_140,
    "replace_uniform", 8, 0, 1, 1, 407
}
```

12.1.4.146 static KeyWord kw_142 () [static]
Initial value:

```
{
    "metric_tracker", 8, 0, 1, 1, 331, 0, 0., 0., 0., 0,
    "num_generations", 0x29, 0, 3, 0, 335, 0, 0., 0., 0., 0,
    "percent_change", 10, 0, 2, 0, 333, 0, 0., 0., 0., 0
}
```
12.1.4.147 static KeyWord kw_143 ()  [static]

Initial value:

{
    {"domination_count",8,0,1,1,305},
    {"layer_rank",8,0,1,1,303}
}

12.1.4.148 static KeyWord kw_144 ()  [static]

Initial value:

{
    {"num_designs",0x29,0,1,0,327,0,2.,0.,0.,0,"{Number designs to keep for max_designs nicher} MethodCommands.html#MethodJEGAMOGA"}
}

12.1.4.149 static KeyWord kw_145 ()  [static]

Initial value:

{
    {"distance",14,0,1,1,323},
    {"max_designs",14,1,1,1,325,kw_144},
    {"radial",14,0,1,1,321}
}

12.1.4.150 static KeyWord kw_146 ()  [static]

Initial value:

{
    {"orthogonal_distance",14,0,1,1,339,0,0.,0.,0.,0,"{Post_processor distance} MethodCommands.html#MethodJEGAMOGA"}
}

12.1.4.151 static KeyWord kw_147 ()  [static]

Initial value:

{
    {"shrinkage_fraction",10,0,1,0,317},
    {"shrinkage_percentage",2,0,1,0,316}
}
12.1.4.152 static KeyWord kw_148 () [static]

Initial value:

```
{"below_limit",10,2,1,1,315,kw_147,0.,0.,0.,0,"{Below limit selec
tion} MethodCommands.html#MethodJEGADC"},
{"elitist",8,0,1,1,309},
{"roulette_wheel",8,0,1,1,311},
{"unique_roulette_wheel",8,0,1,1,313}
```

12.1.4.153 static KeyWord kw_149 () [static]

Initial value:

```
{"convergence_type",8,3,4,0,329,kw_142},
{"crossover_type",8,5,19,0,387,kw_138,0.,0.,0.,0,"{Crossover type} Met
hodCommands.html#MethodJEGADC"},
{"fitness_type",8,2,1,0,301,kw_143,0.,0.,0.,0,"{Fitness type} Met
hodCommands.html#MethodJEGAMOGA"},
{"initialization_type",8,3,18,0,379,kw_139,0.,0.,0.,0,"{Initializ
ation type} MethodCommands.html#MethodJEGADC"},
{"linear_equality_constraint_matrix",14,0,11,0,433,0,0.,0.,0.,0,"{Linea
er equality coefficient matrix} MethodCommands.html#MethodIndControl"},
{"linear_equality_scale_types",15,0,13,0,435,0,0.,0.,0.,0,"{Linea
er equality scaling types} MethodCommands.html#MethodIndControl"},
{"linear_equality_scales",14,0,14,0,437,0,0.,0.,0.,0,"{Linear equ
ality scales} MethodCommands.html#MethodIndControl"},
{"linear_equality_targets",14,0,12,0,433,0,0.,0.,0.,0,"{Linear equ
ality targets} MethodCommands.html#MethodIndControl"},
{"linear_inequality_constraint_matrix",14,0,6,0,421,0,0.,0.,0.,0,
"{Linear inequality coefficient matrix} MethodCommands.html#MethodIndControl"},
{"linear_inequality_lower_bounds",14,0,7,0,423,0,0.,0.,0.,0,"{Lin
ear inequality lower bounds} MethodCommands.html#MethodIndControl"},
{"linear_inequality_scale_types",15,0,9,0,427,0,0.,0.,0.,0,"{Linea
ear inequality scaling types} MethodCommands.html#MethodIndControl"},
{"linear_inequality_scales",14,0,10,0,429,0,0.,0.,0.,0,"{Linear ine
quality scales} MethodCommands.html#MethodIndControl"},
{"linear_inequality_upper_bounds",14,0,8,0,425,0,0.,0.,0.,0,"{Lin
ear inequality upper bounds} MethodCommands.html#MethodIndControl"},
{"logfile",11,0,16,0,375,0,0.,0.,0.,0,"{Log file} MethodCommands
.html#MethodJEGADC"},
{"mutation_type",8,6,20,0,403,kw_141,0.,0.,0.,0,"{Mutation type} Met
hodCommands.html#MethodJEGADC"},
{"niching_type",8,3,3,0,319,kw_145,0.,0.,0.,0,"{Niche pressure ty
pe} MethodCommands.html#MethodJEGAMOGA"},
{"population_size",0x29,0,15,0,373,0,0.,0.,0.,0,"{Number of popul
ation members} MethodCommands.html#MethodJEGADC"},
{"postprocessor_type",8,1,5,0,337,kw_146,0.,0.,0.,0,"{Post proces
 sor type} MethodCommands.html#MethodJEGAMOGA"},
{"print_each_pop",8,0,17,0,377,0,0.,0.,0.,0,"{Population output} Met
hodCommands.html#MethodJEGADC"},
{"replacement_type",8,4,2,0,307,kw_148,0.,0.,0.,0,"{Replacement t
ype} MethodCommands.html#MethodJEGAMOGA"},
{"seed",0x19,0,21,0,419,0,0.,0.,0.,0,"{Random seed} MethodComm
d.html#MethodNonDMC"}
12.1.4.154  static KeyWord kw_150 ()  [static]

Initial value:

{
    {"partitions",
    13, 0, 1, 1, 1531, 0, 0, 0, 0, 0, 0, "{Partitions per variable}
               MethodCommands.html#MethodPSMPS"}
}

12.1.4.155  static KeyWord kw_151 ()  [static]

Initial value:

{
    {"min_boxsize_limit",
    10, 0, 2, 0, 1311, 0, 0, 0, 0, 0, 0, "{Min boxsize limit}
               MethodCommands.html#MethodNCSUDC"},
    {"solution_accuracy",
    2, 0, 1, 0, 1308},
    {"solution_target",
    10, 0, 1, 0, 1309, 0, 0, 0, 0, 0, 0, "{Solution Target}
               MethodCommands.html#MethodNCSUDC"},
    {"volume_boxsize_limit",
    10, 0, 3, 0, 1313, 0, 0, 0, 0, 0, 0, "{Volume boxsize limit}
               MethodCommands.html#MethodNCSUDC"}
}

12.1.4.156  static KeyWord kw_152 ()  [static]

Initial value:

{
    {"absolute_conv_tol",
    10, 0, 2, 0, 583, 0, 0, 0, 0, 0, 0, "{Absolute function convergence tolerance}
               MethodCommands.html#MethodLSNL2SOL"},
    {"covariance",
    9, 0, 0, 0, 595, 0, 0, 0, 0, 0, 0, "{Covariance post-processing}
               MethodCommands.html#MethodLSNL2SOL"},
    {"false_conv_tol",
    10, 0, 6, 0, 591, 0, 0, 0, 0, 0, 0, "{False convergence tolerance}
               MethodCommands.html#MethodLSNL2SOL"},
    {"function_precision",
    10, 0, 1, 0, 581},
    {"initial_trust_radius",
    10, 0, 7, 0, 593, 0, 0, 0, 0, 0, 0, "{Initial trust region radius}
               MethodCommands.html#MethodLSNL2SOL"},
    {"regression_diagnostics",
    8, 0, 9, 0, 597, 0, 0, 0, 0, 0, 0, "{Regression diagnostics post-processing}
               MethodCommands.html#MethodLSNL2SOL"},
    {"singular_conv_tol",
    10, 0, 4, 0, 587, 0, 0, 0, 0, 0, 0, "{Singular convergence tolerance}
               MethodCommands.html#MethodLSNL2SOL"},
    {"singular_radius",
    10, 0, 5, 0, 589, 0, 0, 0, 0, 0, 0, "{Step limit for active constraints}
               MethodCommands.html#MethodLSNL2SOL"},
    {"x_conv_tol",
    10, 0, 3, 0, 585, 0, 0, 0, 0, 0, 0, "{Convergence tolerance for change in parameter vector}
               MethodCommands.html#MethodLSNL2SOL"}
}

12.1.4.157  static KeyWord kw_153 ()  [static]

Initial value:

{
    {"parallel",
    8, 0, 1, 1, 1021},
    {"series",
    8, 0, 1, 1, 1019}
}
12.1.4.158  static KeyWord kw_154 () [static]
Initial value:

```
{
    "gen_reliabilities",8,0,1,1,1015,
    "probabilities",8,0,1,1,1013,
    "system",8,2,2,0,1017,kw_153
}
```

12.1.4.159  static KeyWord kw_155 () [static]
Initial value:

```
{
    "compute",8,3,2,0,1011,kw_154,
    "num_response_levels",15,0,1,0,1009
}
```

12.1.4.160  static KeyWord kw_156 () [static]
Initial value:

```
{
    "distribution",8,2,4,0,1075,kw_15,0.,0.,0.,"\{Distribution type
}MethodCommands.html\#MethodNonD"),
    "gen_reliability_levels",14,1,6,0,1085,kw_16,0.,0.,0.,"\{Generalized
reliability levels\}MethodCommands.html\#MethodNonD"),
    "probability_levels",14,1,5,0,1081,kw_17,0.,0.,0.,"\{Probability
levels\}MethodCommands.html\#MethodNonD"),
    "response_levels",14,2,1,0,1007,kw_155,
    "rng",8,2,7,0,1089,kw_18,0.,0.,0.,"\{Random number generator\}M
ethodCommands.html\#MethodNonDMC"),
    "samples",9,0,3,0,1303,0.,0.,0.,"\{Number of samples\}MethodC
ommands.html\#MethodNonDMC"),
    "seed",0x19,0,2,0,1305,0.,0.,0.,0.,"\{Random seed\}MethodCommand
s.html\#MethodEG")
}
```

12.1.4.161  static KeyWord kw_157 () [static]
Initial value:

```
{
    "num_reliability_levels",13,0,1,0,875,0.,0.,0.,0.,"\{Number of reliabil
ity levels\}MethodCommands.html\#MethodNonD"
}
```

12.1.4.162  static KeyWord kw_158 () [static]
Initial value:
12.1.4.163 static KeyWord kw_159 () [static]

Initial value:

```
{
    "gen_reliabilities", 8, 0, 1, 1, 887,
    "probabilities", 8, 0, 1, 1, 883,
    "reliabilities", 8, 0, 1, 1, 885,
    "system", 8, 2, 2, 0, 889, kw_158
}
```

12.1.4.164 static KeyWord kw_160 () [static]

Initial value:

```
{
    "compute", 8, 4, 2, 0, 881, kw_159, 0, 0, 0, 0,"\{Target statistics for response levels\} MethodCommands.html#MethodNonD",
    "num_response_levels", 13, 0, 1, 0, 879, 0, 0, 0, 0,"\{Number of response levels\} MethodCommands.html#MethodNonD"
}
```

12.1.4.165 static KeyWord kw_161 () [static]

Initial value:

```
{
    "annotated", 8, 0, 1, 0, 719,
    "freeform", 8, 0, 1, 0, 721
}
```

12.1.4.166 static KeyWord kw_162 () [static]

Initial value:

```
{
    "noise_tolerance", 14, 0, 1, 0, 691
}
```

12.1.4.167 static KeyWord kw_163 () [static]

Initial value:

```
{
    "noise_tolerance", 14, 0, 1, 0, 695
}
```
12.1.4.168 static KeyWord kw_164 () [static]

Initial value:

```
{ "l2_penalty":10,0,2,0,701,0,0,0.0,0.,0,"{l2_penalty used for elastic net modification of LASSO} MethodCommands.html#MethodNonDPCE"},
{"noise_tolerance":14,0,1,0,699}
}
```

12.1.4.169 static KeyWord kw_165 () [static]

Initial value:

```
{ "equality_constrained":8,0,1,0,681},
{"svd",8,0,1,0,679}
}
```

12.1.4.170 static KeyWord kw_166 () [static]

Initial value:

```
{ "noise_tolerance":14,0,1,0,685}
}
```

12.1.4.171 static KeyWord kw_167 () [static]

Initial value:

```
{ "basis_pursuit",8,0,2,0,687,0,0,0,0,0,"{L1 minimization via Basis Pursuit (BP)} MethodCommands.html#MethodNonDPCE"},
{"basis_pursuit_denoising",8,1,2,0,689,kw_162,0,0,0,0,"{L1 minimization via Basis Pursuit DeNoising (BPDN)} MethodCommands.html#MethodNonDPCE"},
{"cross_validation",8,0,3,0,703,0,0,0,0,0,"{Specify whether to use cross validation} MethodCommands.html#MethodNonDPCE"},
{"import_points_file",11,2,7,0,717,kw_161,0,0,0,0,"{File name for points to be imported for forming a PCE (unstructured grid assumed)} MethodCommands.html#MethodNonDPCE"},
{"lars",0,1,2,0,692,kw_163},
{"lasso",0,2,2,0,696,kw_164},
{"least_absolute_shrinkage",8,2,2,0,697,kw_164,0,0,0,0,"{L1 minimization via Least Absolute Shrinkage Operator (LASSO)} MethodCommands.html#MethodNonDPCE"},
{"least_angle_regression",8,1,2,0,693,kw_163,0,0,0,0,"{L1 minimization via Least Angle Regression (LARS)} MethodCommands.html#MethodNonDPCE"},
{"least_squares",8,2,2,0,677,kw_165,0,0,0,0,"{Least squares regression} MethodCommands.html#MethodNonDPCE"},
```
12.1.4.172 static KeyWord kw_168 () [static]

Initial value:

```
{
  "import_points_file",11,2,3,0,717,kw_161,0.,0.,0.,"{File name for points to be imported for forming a PCE (unstructured grid assumed) MethodCommands.html#MethodNonDPCE"},
  "incremental_lhs",8,0,2,0,715,0.,0.,0.,"{Use incremental LHS for expansion_samples} MethodCommands.html#MethodNonDPCE"),
  "reuse_points",8,0,1,0,713,
  "reuse_samples",0,0,1,0,712}
```

12.1.4.173 static KeyWord kw_169 () [static]

Initial value:

```
{
  "collocation_points",13,18,2,1,671,kw_167,0.,0.,0.,"{Number collocation points to estimate coeffs} MethodCommands.html#MethodNonDPCE"},
  "collocation_ratio",10,18,2,1,673,kw_167,0.,0.,0.,"{Collocation point oversampling ratio to estimate coeffs} MethodCommands.html#MethodNonDPCE"},
  "dimension_preference",14,0,1,0,669,
  "expansion_import_file",11,0,2,1,723,0.,0.,0.,"{Import file for PCE coefficients} MethodCommands.html#MethodNonDPCE"),
  "expansion_samples",13,4,2,1,711,kw_168,0.,0.,0.,"{Number simulation samples to estimate coeffs} MethodCommands.html#MethodNonDPCE"}
```

12.1.4.174 static KeyWord kw_170 () [static]

Initial value:

```
{
  "annotated",8,0,1,0,769,
  "freeform",8,0,1,0,771}
```
12.1.4.175 static KeyWord kw_171 () [static]

Initial value:

```
{
  "annotated",8,0,1,0,737,
  "freeform",8,0,1,0,739
}
```

12.1.4.176 static KeyWord kw_172 () [static]

Initial value:

```
{
  "collocation_points",13,0,1,1,727,
  "cross_validation",8,0,2,0,729,
  "import_points_file",11,2,5,0,735,kw_171,0.,0.,0.,0.,"{File name for points to be imported as the basis for the initial emulator} MethodCommands.html#MethodNonDBayesCalib"),
  "reuse_points",8,0,4,0,733),
  "reuse_samples",0,0,4,0,732),
  "tensor_grid",13,0,3,0,731
}
```

12.1.4.177 static KeyWord kw_173 () [static]

Initial value:

```
{
  "decay",8,0,1,1,643),
  "generalized",8,0,1,1,645),
  "sobol",8,0,1,1,641
}
```

12.1.4.178 static KeyWord kw_174 () [static]

Initial value:

```
{
  "dimension_adaptive",8,3,1,1,639,kw_173),
  "uniform",8,0,1,1,637
}
```

12.1.4.179 static KeyWord kw_175 () [static]

Initial value:

```
{
  "dimension_preference",14,0,1,0,659,0,0.,0.,0.,0.,"{Dimension preference for anisotropic tensor and sparse grids} MethodCommands.html#MethodNonDPC
```
12.1.4.180  static KeyWord kw_176 () [static]  
Initial value:

    {
      "adapt_import",8,0,1,1,763,
      "import",8,0,1,1,761,
      "mm_adapt_import",8,0,1,1,765
    }

12.1.4.181  static KeyWord kw_177 () [static]  
Initial value:

    {
      "lhs",8,0,1,1,755,
      "random",8,0,1,1,757
    }

12.1.4.182  static KeyWord kw_178 () [static]  
Initial value:

    {
      "dimension_preference",14,0,2,0,659,0,0,0,0,0,0,"{Dimension preference for anisotropic tensor and sparse grids} MethodCommands.html#MethodNonDPC E*
      "nested",8,0,3,0,661,
      "non_nested",8,0,3,0,663,
      "restricted",8,0,1,0,655,
      "unrestricted",8,0,1,0,657
    }

12.1.4.183  static KeyWord kw_179 () [static]  
Initial value:

    {
      "drop_tolerance",10,0,2,0,745,0,0,0,0,0,0,"{VBD tolerance for omitting small indices} MethodCommands.html#MethodNonDMC*
      "interaction_order",0x19,0,1,0,743,0,0,0,0,0,0,"{Restriction of order of VBD iterations} MethodCommands.html#MethodNonDPCE*
    }
12.1 Dakota Namespace Reference

12.1.4.184  static KeyWord kw_180 () [static]

Initial value:

```
{
    {"askey", 8, 0, 2, 0, 647},
    {"cubature_integrand", 8, 0, 3, 1, 665, 0, 0, 0, 0, 0, "{Cubature integrand for PCE coefficient estimation} MethodCommands.html#MethodNonDPCE"},
    {"diagonal_covariance", 8, 0, 5, 0, 747},
    {"distribution", 8, 2, 12, 0, 1075, kw_15, 0, 0, 0, 0, "{Distribution type} MethodCommands.html#MethodNonD"},
    {"expansion_order", 13, 5, 3, 1, 667, kw_169, 0, 0, 0, 0, "{Expansion order} MethodCommands.html#MethodNonDPCE"},
    {"export_points_file", 11, 2, 9, 0, 767, kw_170, 0, 0, 0, 0, "{File name for exporting approximation-based samples from evaluating the PCE} MethodCommands.html#MethodNonDPCE"},
    {"fixed_seed", 8, 0, 18, 0, 871, 0, 0, 0, 0, 0, "{Fixed seed flag} MethodCommands.html#MethodNonD"},
    {"full_covariance", 8, 0, 5, 0, 749},
    {"gen_reliability_levels", 14, 1, 14, 0, 1085, kw_16, 0, 0, 0, 0, "{Generalized reliability levels} MethodCommands.html#MethodNonD"},
    {"least_interpolation", 0, 6, 3, 1, 724, kw_172},
    {"normalized", 8, 0, 6, 0, 751, 0, 0, 0, 0, 0, "{Output PCE coefficients corresponding to normalized basis} MethodCommands.html#MethodNonDPCE"},
    {"oli", 0, 6, 3, 1, 724, kw_172},
    {"orthogonal_least_interpolation", 8, 6, 3, 1, 725, kw_172, 0, 0, 0, 0, 0, "{Orthogonal Least Interpolation (OLI)} MethodCommands.html#MethodNonDPCE"},
    {"p_refinement", 8, 2, 1, 0, 635, kw_174, 0, 0, 0, 0, 0, "{Automated polynomial order refinement} MethodCommands.html#MethodNonDPCE"},
    {"probability_levels", 14, 1, 13, 0, 1081, kw_17, 0, 0, 0, 0, 0, "{Probability levels} MethodCommands.html#MethodNonD"},
    {"quadrature_order", 13, 3, 3, 1, 651, kw_175, 0, 0, 0, 0, 0, "{Quadrature order for PCE coefficient estimation} MethodCommands.html#MethodNonD"},
    {"reliability_levels", 14, 1, 16, 0, 873, kw_157, 0, 0, 0, 0, 0, "{Reliability levels} MethodCommands.html#MethodNonD"},
    {"response_levels", 14, 2, 17, 0, 877, kw_160, 0, 0, 0, 0, 0, "{Response levels} MethodCommands.html#MethodNonD"},
    {"rng", 8, 2, 15, 0, 1089, kw_18, 0, 0, 0, 0, 0, "{Random number generator} MethodCommands.html#MethodNonD"},
    {"sample_refinement", 8, 3, 8, 0, 759, kw_176, 0, 0, 0, 0, 0, "{Importance sampling refinement} MethodCommands.html#MethodNonDPCE"},
    {"sample_type", 8, 2, 7, 0, 753, kw_177, 0, 0, 0, 0, 0, "{Sampling type} MethodCommands.html#MethodNonD"},
    {"samples", 9, 0, 11, 0, 1303, 0, 0, 0, 0, 0, "{Number of samples} MethodCommands.html#MethodNonD"},
    {"seed", 0x19, 0, 10, 0, 1305, 0, 0, 0, 0, 0, "{Random seed} MethodCommands.html#MethodNonD"},
    {"sparse_grid_level", 13, 5, 3, 1, 653, kw_178, 0, 0, 0, 0, 0, "{Sparse grid level for PCE coefficient estimation} MethodCommands.html#MethodNonDPCE"},
    {"variance_based_decomp", 8, 2, 4, 0, 741, kw_179, 0, 0, 0, 0, 0, "{Variance based decomposition (VBD)} MethodCommands.html#MethodNonD"},
    {"wiener", 8, 0, 2, 0, 649}
}

```

12.1.4.185  static KeyWord kw_181 () [static]

Initial value:

```
{
}

```
12.1.4.186 static Keyword kw_182 () [static]

Initial value:

```
{
    {"incremental_lhs", 8, 1, 1, 1861, kw_181},
    {"incremental_random", 8, 1, 1, 1861, kw_181},
    {"lhs", 8, 0, 1, 1859},
    {"random", 8, 0, 1, 1857}
}
```

12.1.4.187 static Keyword kw_183 () [static]

Initial value:

```
{
    {"drop_tolerance", 10, 0, 1, 0, 1869}
}
```

12.1.4.188 static Keyword kw_184 () [static]

Initial value:

```
{
    {"distribution", 8, 2, 5, 0, 1075, kw_15, 0, 0, 0, 0,"{Distribution type
        } MethodCommands.html#MethodNonDMC"},
    {"fixed_seed", 8, 0, 11, 1, 871, 0, 0, 0, 0,"{Fixed seed flag} Method
        Commands.html#MethodNonDMC"},
    {"gen_reliability_levels", 14, 1, 7, 0, 1085, kw_16, 0, 0, 0, 0,"{Genera
        lized reliability levels} MethodCommands.html#MethodNonD"},
    {"probability_levels", 14, 1, 6, 0, 1081, kw_17, 0, 0, 0, 0,"{Probabilit
        y levels} MethodCommands.html#MethodNonD"},
    {"reliability_levels", 14, 1, 9, 0, 873, kw_157, 0, 0, 0, 0,"{Reliability
        levels} MethodCommands.html#MethodNonD"},
    {"response_levels", 14, 2, 10, 0, 877, kw_160, 0, 0, 0, 0,"{Response lev
        els} MethodCommands.html#MethodNonD"},
    {"rng", 8, 2, 8, 0, 1089, kw_18, 0, 0, 0, 0,"{Random number generator} M
        ethodCommands.html#MethodNonDMC"},
    {"sample_type", 8, 4, 1, 0, 855, kw_182},
    {"samples", 9, 0, 4, 0, 1303, 0, 0, 0, 0,"{Number of samples} MethodC
        ommands.html#MethodNonDMC"},
    {"seed", 0x19, 0, 3, 0, 1305, 0, 0, 0, 0,"{Random seed} MethodCommand
        s.html#MethodEG"},
    {"variance_based_decomp", 8, 1, 2, 0, 867, kw_183}
}
```
12.1.4.189  static KeyWord kw_185 ()  [static]

Initial value:

{  
      {"annotated",8,0,1,0,849},
      {"freeform",8,0,1,0,851}
    }

12.1.4.190  static KeyWord kw_186 ()  [static]

Initial value:

{  
      {"generalized",8,0,1,1,793},
      {"sobol",8,0,1,1,791}
    }

12.1.4.191  static KeyWord kw_187 ()  [static]

Initial value:

{  
      {"dimension_adaptive",8,2,1,1,789,kw_186},
      {"local_adaptive",8,0,1,1,795},
      {"uniform",8,0,1,1,787}
    }

12.1.4.192  static KeyWord kw_188 ()  [static]

Initial value:

{  
      {"generalized",8,0,1,1,783},
      {"sobol",8,0,1,1,781}
    }

12.1.4.193  static KeyWord kw_189 ()  [static]

Initial value:

{  
      {"dimension_adaptive",8,2,1,1,779,kw_188},
      {"uniform",8,0,1,1,777}
    }
12.1.4.194 static KeyWord kw_190 () [static]

Initial value:

```cpp
{
    {"adapt_import",8,0,1,1,843},
    {"import",8,0,1,1,841},
    {"mm_adapt_import",8,0,1,1,845}
}
```

12.1.4.195 static KeyWord kw_191 () [static]

Initial value:

```cpp
{
    {"lhs",8,0,1,1,835},
    {"random",8,0,1,1,837}
}
```

12.1.4.196 static KeyWord kw_192 () [static]

Initial value:

```cpp
{
    {"hierarchical",8,0,2,0,813},
    {"nodal",8,0,2,0,811},
    {"restricted",8,0,1,0,807},
    {"unrestricted",8,0,1,0,809}
}
```

12.1.4.197 static KeyWord kw_193 () [static]

Initial value:

```cpp
{
    {"drop_tolerance",10,0,2,0,827,0,0,,0,0,"{VBD tolerance for omitting small indices} MethodCommands.html#MethodNonDSC"},
    {"interaction_order",0x19,0,1,0,825,0,0,,0,0,"{Restriction of order of VBD iterations} MethodCommands.html#MethodNonDSC"}
}
```

12.1.4.198 static KeyWord kw_194 () [static]

Initial value:

```cpp
{
    {"askey",8,0,2,0,799},
    {"diagonal_covariance",8,0,8,0,829},
    {"dimension_preference",14,0,4,0,815,0,0,,0,0,"{Dimension preference for anisotropic tensor and sparse grids} MethodCommands.html#MethodNonDSC
```
12.1 Dakota Namespace Reference

12.1.4.199 static KeyWord kw_195 () [static]

Initial value:

```cpp
{
    "misc_options",15,0,1,0,601
}
```

12.1.4.200 static KeyWord kw_196 () [static]

Initial value:

```cpp
{
    "function_precision",10,0,11,0,203,0,0,0,0,0,0,0,"{Function precision} MethodCommands.html#MethodNPSOLDC",
    "linear_equality_constraint_matrix",14,0,6,0,431,0,0,0,0,0,0,0,0,"{Linear equality constraint matrix} MethodCommands.html#MethodNPSOLDC",
    "non_nested",8,0,6,0,821,
    "probability_levels",14,1,15,0,1081,kw_189,0,0,0,0,0,0,"{Probability levels} MethodCommands.html#MethodNPSOLDC",
    "reliability_levels",14,1,18,0,873,kw_190,0,0,0,0,0,0,"{Reliability levels} MethodCommands.html#MethodNPSOLDC",
    "response_levels",14,2,19,0,877,kw_191,0,0,0,0,0,0,"{Response levels} MethodCommands.html#MethodNPSOLDC",
    "rng",8,2,17,0,1089,kw_187,0,0,0,0,0,0,"{Random number generator} MethodCommands.html#MethodNPSOLDC",
    "sample_refinement",8,3,10,0,839,kw_192,0,0,0,0,0,0,"{Sample refinement} MethodCommands.html#MethodNPSOLDC",
    "sample_type",8,2,9,0,833,kw_193,0,0,0,0,0,0,"{Sample type} MethodCommands.html#MethodNPSOLDC",
    "samples",9,0,13,0,1303,0,0,0,0,0,0,"{Number of samples} MethodCommands.html#MethodNPSOLDC",
    "seed",0x19,0,12,0,1305,0,0,0,0,0,0,"{Random seed} MethodCommands.html#MethodNPSOLDC",
    "sparse_grid_level",13,4,3,1,805,kw_194,0,0,0,0,0,0,"{Sparse grid level for collocation points} MethodCommands.html#MethodNPSOLDC",
    "use_derivatives",8,0,5,0,817,0,0,0,0,0,0,"{Use derivatives} MethodCommands.html#MethodNPSOLDC",
    "variance_based_decomp",8,2,7,0,823,kw_195,0,0,0,0,0,0,"{Variance-based decomposition (VBD)} MethodCommands.html#MethodNPSOLDC",
    "wiener",8,0,2,0,801,
}
Linear equality coefficient matrix} MethodCommands.html#MethodIndControl",
   {"linear_equality_scale_types",15,0,8,0,435,0,0,0,0,0,"{Linear
   equality scaling types} MethodCommands.html#MethodIndControl"},
   {"linear_equality_scales",14,0,9,0,437,0,0,0,0,0,"{Linear equa
   lity scales} MethodCommands.html#MethodIndControl"},
   {"linear_equality_targets",14,0,7,0,433,0,0,0,0,0,"{Linear equa
   lity targets} MethodCommands.html#MethodIndControl"},
   {"linear_inequality_constraint_matrix",14,0,1,0,421,0,0,0,0,0,
   "{Linear inequality coefficient matrix} MethodCommands.html#MethodIndControl"},
   {"linear_inequality_lower_bounds",14,0,2,0,423,0,0,0,0,0,"{Lin
   ear inequality lower bounds} MethodCommands.html#MethodIndControl"},
   {"linear_inequality_scale_types",15,0,4,0,427,0,0,0,0,0,"{Line
   ar inequality scaling types} MethodCommands.html#MethodIndControl"},
   {"linear_inequality_scales",14,0,5,0,429,0,0,0,0,0,"{Linear in
   equality scales} MethodCommands.html#MethodIndControl"},
   {"linear_inequality_targets",14,0,3,0,425,0,0,0,0,0,"{Linear ine
   quality targets} MethodCommands.html#MethodIndControl"},
   {"linesearch_tolerance",10,0,12,0,205,0,0,0,0,0,"{Line search
tolerance} MethodCommands.html#MethodNPSOLDC"},
   {"verify_level",9,0,10,0,201,0,0,0,0,0,"{Gradient verification
level} MethodCommands.html#MethodNPSOLDC"}
}

12.1.4.201 static Keyword kw_197 () [static]

Initial value:

{"gradient_tolerance",10,0,11,0,243},
{"linear_equality_constraint_matrix",14,0,6,0,431,0,0,0,0,0,"{Lin
ear equality coefficient matrix} MethodCommands.html#MethodIndControl"},
{"linear_equality_scale_types",15,0,8,0,435,0,0,0,0,0,"{Linear
equality scaling types} MethodCommands.html#MethodIndControl"},
{"linear_equality_scales",14,0,9,0,437,0,0,0,0,0,"{Linear equa
lity scales} MethodCommands.html#MethodIndControl"},
{"linear_equality_targets",14,0,7,0,433,0,0,0,0,0,"{Linear equa
lity targets} MethodCommands.html#MethodIndControl"},
{"linear_inequality_constraint_matrix",14,0,1,0,421,0,0,0,0,0,
"{Linear inequality coefficient matrix} MethodCommands.html#MethodIndControl"},
{"linear_inequality_lower_bounds",14,0,2,0,423,0,0,0,0,0,"{Lin
ear inequality lower bounds} MethodCommands.html#MethodIndControl"},
{"linear_inequality_scale_types",15,0,4,0,427,0,0,0,0,0,"{Line
ar inequality scaling types} MethodCommands.html#MethodIndControl"},
{"linear_inequality_scales",14,0,5,0,429,0,0,0,0,0,"{Linear in
equality scales} MethodCommands.html#MethodIndControl"},
{"linear_inequality_targets",14,0,3,0,425,0,0,0,0,0,"{Linear ine
quality targets} MethodCommands.html#MethodIndControl"},
{"max_step",10,0,10,0,241}
}

12.1.4.202 static Keyword kw_198 () [static]

Initial value:

{"linear_equality_constraint_matrix",14,0,7,0,431,0,0,0,0,0,"{Lin
ear equality coefficient matrix} MethodCommands.html#MethodIndControl"},
{"linear_inequality_constraint_matrix",14,0,1,0,421,0,0,0,0,0,
"{Linear inequality coefficient matrix} MethodCommands.html#MethodIndControl"},
{"linear_inequality_lower_bounds",14,0,2,0,423,0,0,0,0,0,"{Lin
ear inequality lower bounds} MethodCommands.html#MethodIndControl"},
{"linear_inequality_scale_types",15,0,4,0,427,0,0,0,0,0,"{Line
ar inequality scaling types} MethodCommands.html#MethodIndControl"},
{"linear_inequality_scales",14,0,5,0,429,0,0,0,0,0,"{Linear in
equality scales} MethodCommands.html#MethodIndControl"},
{"linear_inequality_upper_bounds",14,0,3,0,425,0,0,0,0,0,"{Lin
ear inequality upper bounds} MethodCommands.html#MethodIndControl"},
{"linesearch_tolerance",10,0,12,0,205,0,0,0,0,0,"{Line search
tolerance} MethodCommands.html#MethodNPSOLDC"},
{"verify_level",9,0,10,0,201,0,0,0,0,0,"{Gradient verification
level} MethodCommands.html#MethodNPSOLDC"}
12.1.4.203 static KeyWord kw_199 () [static]

Initial value:

```
{ "argaez_tapia", 8, 0, 1, 1, 233 },
{ "el_bakry", 8, 0, 1, 1, 231 },
{ "van_shanno", 8, 0, 1, 1, 235 }
```

12.1.4.204 static KeyWord kw_200 () [static]

Initial value:

```
{ "gradient_based_line_search", 8, 0, 1, 1, 223, 0, 0, 0, 0, 0, 0, "[CHOOSE line search type]" },
{ "tr_pds", 8, 0, 1, 1, 227 },
{ "trust_region", 8, 0, 1, 1, 225 },
{ "value_based_line_search", 8, 0, 1, 1, 221 }
```

12.1.4.205 static KeyWord kw_201 () [static]

Initial value:

```
{ "centering_parameter", 10, 0, 4, 0, 239 },
{ "gradient_tolerance", 10, 0, 15, 0, 243 },
{ "linear_equality_constraint_matrix", 14, 0, 10, 0, 433, 0, 0, 0, 0, 0, 0, "[Linear equality constraint matrix]" },
{ "linear_equality_scale_types", 15, 0, 12, 0, 435, 0, 0, 0, 0, 0, 0, "[Linear equality scaling types]" },
{ "linear_equality_scales", 14, 0, 13, 0, 437, 0, 0, 0, 0, 0, 0, "[Linear equality scales]" }
```
12.1.4.206 static KeyWord kw_202 () [static]

Initial value:

```
{"debug",8,0,1,1,73,0,0,0,0,0,"[CHOOSE output level]",
{"normal",8,0,1,1,77},
{"quiet",8,0,1,1,79},
{"silent",8,0,1,1,81},
{"verbose",8,0,1,1,75}
}
```

12.1.4.207 static KeyWord kw_203 () [static]

Initial value:

```
{"partitions",13,0,1,0,1301,0,0,0,0,0,"{Number of partitions} MethodCommands.html#MethodPSUADE"},
{"samples",9,0,3,0,1303,0,0,0,0,0,"{Number of samples} MethodCommands.html#MethodNonDMC"},
{"seed",0x19,0,2,0,1305,0,0,0,0,0,"{Random seed} MethodCommands.html#MethodEG"}
}
```

12.1.4.208 static KeyWord kw_204 () [static]

Initial value:

```
{"converge_order",8,0,1,1,1537},
{"converge_qoi",8,0,1,1,1539},
{"estimate_order",8,0,1,1,1535},
{"refinement_rate",10,0,2,0,1541,0,0,0,0,0,"{Refinement rate} MethodCommands.html#MethodSolnRichardson"} }
```
12.1.4.209 static KeyWord kw_205 () [static]

Initial value:

{
    "num_generations",0x29,0,2,0,371,
    "percent_change",10,0,1,0,369
}

12.1.4.210 static KeyWord kw_206 () [static]

Initial value:

{
    "num_generations",0x29,0,2,0,365,0,0..0.,0.,0.,"{Number of generations (for convergence test) } MethodCommands.html#MethodJEGASOGA"),
    "percent_change",10,0,1,0,363,0,0..0.,0.,"{Percent change in fitness} MethodCommands.html#MethodJEGASOGA"
}

12.1.4.211 static KeyWord kw_207 () [static]

Initial value:

{
    "average_fitness_tracker",8,2,1,1,367,kw_205,
    "best_fitness_tracker",8,2,1,1,361,kw_206
}

12.1.4.212 static KeyWord kw_208 () [static]

Initial value:

{
    "constraint_penalty",10,0,2,0,347,0,0..0.,0.,0.,"{Constraint penalty in merit function} MethodCommands.html#MethodJEGASOGA"),
    "merit_function",8,0,1,1,345
}

12.1.4.213 static KeyWord kw_209 () [static]

Initial value:

{
    "elitist",8,0,1,1,351,
    "favor_feasible",8,0,1,1,353,
    "roulette_wheel",8,0,1,1,355,
    "unique_roulette_wheel",8,0,1,1,357
}
12.1.4.214 static KeyWord kw_210 () [static]

Initial value:

```c
{
    {"convergence_type",8,2,3,0,359,kw_207,0,0,0,0,"{Convergence type} MethodCommands.html#MethodJEGASOGA"},
    {"crossover_type",8,5,17,0,387,kw_138,0,0,0,0,"{Crossover type} MethodCommands.html#MethodJEGADCC"},
    {"fitness_type",8,2,1,0,343,kw_208,0,0,0,0,"{Fitness type} MethodCommands.html#MethodJEGASOGA"},
    {"initialization_type",8,3,16,0,379,kw_139,0,0,0,0,"{Initialization type} MethodCommands.html#MethodJEGADC"},
    {"linear_equality_constraint_matrix",14,0,9,0,431,0,0,0,0,0,"{Linear equality constraint matrix} MethodCommands.html#MethodIndControl"},
    {"linear_equality_scale_types",15,0,11,0,435,0,0,0,0,0,"{Linear equality scaling types} MethodCommands.html#MethodIndControl"},
    {"linear_equality_scales",14,0,12,0,437,0,0,0,0,0,"{Linear equality scales} MethodCommands.html#MethodIndControl"},
    {"linear_equality_targets",14,0,10,0,433,0,0,0,0,0,"{Linear equality targets} MethodCommands.html#MethodIndControl"},
    {"linear_inequality_constraint_matrix",14,0,4,0,421,0,0,0,0,0,"{Linear inequality constraint matrix} MethodCommands.html#MethodIndControl"},
    {"linear_inequality_scale_types",15,0,7,0,427,0,0,0,0,0,"{Linear inequality scaling types} MethodCommands.html#MethodIndControl"},
    {"linear_inequality_scales",14,0,8,0,429,0,0,0,0,0,"{Linear inequality scales} MethodCommands.html#MethodIndControl"},
    {"linear_inequality_targets",14,0,6,0,425,0,0,0,0,0,"{Linear inequality targets} MethodCommands.html#MethodIndControl"},
    {"log_file",11,0,14,0,375,0,0,0,0,0,"{Log file} MethodCommands.html#MethodJEGADC"},
    {"mutation_type",8,6,18,0,403,kw_141,0,0,0,0,"{Mutation type} MethodCommands.html#MethodJEGADDC"},
    {"population_size",0x29,0,13,0,373,0,0,0,0,0,"{Number of population members} MethodCommands.html#MethodJEGADC"},
    {"print_each_pop",8,0,15,0,377,0,0,0,0,0,"{Population output} MethodCommands.html#MethodJEGADC"},
    {"replacement_type",8,4,2,0,349,kw_209,0,0,0,0,0,"{Replacement type} MethodCommands.html#MethodJEGASOGA"},
    {"seed",0x19,0,19,0,419,0,0,0,0,0,"{Random seed} MethodCommands.html#MethodNonDMC"}
}
```

12.1.4.215 static KeyWord kw_211 () [static]

Initial value:

```c
{
    {"function_precision",10,0,12,0,203,0,0,0,0,0,"{Function precision} MethodCommands.html#MethodNPSOLDC"},
    {"linear_equality_constraint_matrix",14,0,7,0,431,0,0,0,0,0,"{Linear equality constraint matrix} MethodCommands.html#MethodIndControl"},
    {"linear_equality_scale_types",15,0,9,0,435,0,0,0,0,0,"{Linear equality scaling types} MethodCommands.html#MethodIndControl"},
    {"linear_equality_scales",14,0,10,0,437,0,0,0,0,0,"{Linear equality scales} MethodCommands.html#MethodIndControl"},
    {"linear_equality_targets",14,0,8,0,433,0,0,0,0,0,"{Linear equality targets} MethodCommands.html#MethodIndControl"},
    {"log_file",11,0,14,0,375,0,0,0,0,0,"{Log file} MethodCommands.html#MethodJEGADC"},
    {"mutation_type",8,6,18,0,403,kw_141,0,0,0,0,"{Mutation type} MethodCommands.html#MethodJEGADDC"},
    {"population_size",0x29,0,13,0,373,0,0,0,0,0,"{Number of population members} MethodCommands.html#MethodJEGADC"},
    {"print_each_pop",8,0,15,0,377,0,0,0,0,0,"{Population output} MethodCommands.html#MethodJEGADC"},
    {"replacement_type",8,4,2,0,349,kw_209,0,0,0,0,0,"{Replacement type} MethodCommands.html#MethodJEGASOGA"},
    {"seed",0x19,0,19,0,419,0,0,0,0,0,"{Random seed} MethodCommands.html#MethodNonDMC"}
}
```
12.1 Dakota Namespace Reference

12.1.4.216 static KeyWord kw_212 () [static]

Initial value:

```
{ "approx_method_name", 11, 0, 1, 1, 605, 0, 0, 0, 0, 0, 0, "[CHOOSE sub-method ref.] [Approximate sub-problem minimization method name] MethodCommands.html#MethodSBG" },
{ "approx_method_pointer", 11, 0, 1, 1, 607, 0, 0, 0, 0, 0, 0, "[Approximate sub-problem minimization method pointer] MethodCommands.html#MethodSBG" },
{ "replace_points", 8, 0, 2, 0, 609, 0, 0, 0, 0, 0, 0, "[Replace points used in surrogate construction with best points from previous iteration] MethodCommands.html#MethodSBG" }
```

12.1.4.217 static KeyWord kw_213 () [static]

Initial value:

```
{ "filter", 8, 0, 1, 1, 151, 0, 0, 0, 0, 0, 0, 0, "@ [CHOOSE acceptance logic]" },
{ "tr_ratio", 8, 0, 1, 1, 149 }
```

12.1.4.218 static KeyWord kw_214 () [static]

Initial value:

```
{ "augmented_lagrangian_objective", 8, 0, 1, 1, 127, 0, 0, 0, 0, 0, 0, 0, "[CHOOSE objective formulation]" },
{ "lagrangian_objective", 8, 0, 1, 1, 129 },
{ "linearized_constraints", 8, 0, 2, 2, 133, 0, 0, 0, 0, 0, 0, "[CHOOSE constraint formulation]" }
```
12.1.4.219  static KeyWord kw_215 ()  [static]

Initial value:

```
{"homotopy",8,0,1,1,155}
```

12.1.4.220 static KeyWord kw_216 ()  [static]

Initial value:

```
{"adaptive_penalty_merit",8,0,1,1,141,0,0.,0.,0.,0,"[CHOOSE merit
function]",
"augmented_lagrangian_merit",8,0,1,1,145,0,0.,0.,0.,0,"@"},
{"lagrangian_merit",8,0,1,1,143},
{"penalty_merit",8,0,1,1,139}
```

12.1.4.221 static KeyWord kw_217  [static]

Initial value:

```
{"contract_threshold",10,0,3,0,113,0,0.,0.,0.,0,"{Shrink trust re
region if trust region ratio is below this value} MethodCommands.html#MethodSBL"},
{"contraction_factor",10,0,5,0,117,0,0.,0.,0.,0,"{Trust region con
traction factor} MethodCommands.html#MethodSBL"},
{"expand_threshold",10,0,4,0,115,0,0.,0.,0.,0,"{Expand trust regi
on if trust region ratio is above this value} MethodCommands.html#MethodSBL"},
{"expansion_factor",10,0,6,0,119,0,0.,0.,0.,0,"{Trust region expa
nsion factor} MethodCommands.html#MethodSBL"},
{"initial_size",10,0,1,0,109,0,0.,0.,0.,0,"{Trust region initial
size (relative to bounds)} MethodCommands.html#MethodSBL"},
{"minimum_size",10,0,2,0,111,0,0.,0.,0.,0,"{Trust region minimum
size} MethodCommands.html#MethodSBL"}
```

12.1.4.222 static KeyWord kw_218 ()  [static]

Initial value:

```
{"acceptance_logic",8,2,7,0,147,kw_213,0.,0.,0.,0,"{SBL iterate a
```
**12.1 Dakota Namespace Reference**

12.1.4.223 static KeyWord kw_219 () [static]

**Initial value:**

```cpp

{"final_point",14,0,1,1,1507,0,0,0,0,0,0,"[CHOOSE final pt or increment] Termination point of vector} MethodCommands.html#MethodPSVPS"},
{"num_steps",9,0,2,2,1511,0,0,0,0,0,"[Number of steps along vector] Vector commands.html#MethodPSVPS"},
{"step_vector",14,0,1,1,1509,0,0,0,0,0,"[Step vector] Vector commands.html#MethodPSVPS"}
```

12.1.4.224 static KeyWord kw_221 () [static]

**Initial value:**

```cpp

```

---

Dakota Version 5.4 Developers Manual generated on November 7, 2013
202

Namespace Documentation
{"optional_interface_responses_pointer",11,0,1,0,1747,0,0.,0.,0.,
0,"{Responses pointer for nested model optional interfaces} ModelCommands.html#Mo
delNested"}
}

12.1.4.225

static KeyWord kw_222 () [static]

Initial value:
{
{"primary_response_mapping",14,0,3,0,1755,0,0.,0.,0.,0,"{Primary
response mappings for nested models} ModelCommands.html#ModelNested"},
{"primary_variable_mapping",15,0,1,0,1751,0,0.,0.,0.,0,"{Primary
variable mappings for nested models} ModelCommands.html#ModelNested"},
{"secondary_response_mapping",14,0,4,0,1757,0,0.,0.,0.,0,"{Second
ary response mappings for nested models} ModelCommands.html#ModelNested"},
{"secondary_variable_mapping",15,0,2,0,1753,0,0.,0.,0.,0,"{Second
ary variable mappings for nested models} ModelCommands.html#ModelNested"}
}

12.1.4.226

static KeyWord kw_223 () [static]

Initial value:
{
{"optional_interface_pointer",11,1,1,0,1745,kw_221,0.,0.,0.,0,"{O
ptional interface set pointer} ModelCommands.html#ModelNested"},
{"sub_method_pointer",11,4,2,1,1749,kw_222,0.,0.,0.,0,"{Sub-metho
d pointer for nested models} ModelCommands.html#ModelNested"}
}

12.1.4.227

static KeyWord kw_224 () [static]

Initial value:
{
{"interface_pointer",11,0,1,0,1555,0,0.,0.,0.,0,"{Interface set p
ointer} ModelCommands.html#ModelSingle"}
}

12.1.4.228

static KeyWord kw_225 () [static]

Initial value:
{
{"annotated",8,0,1,0,1709},
{"freeform",8,0,1,0,1711}
}

Dakota Version 5.4 Developers Manual generated on November 7, 2013


12.1.4.229 static KeyWord kw_226 () [static]
Initial value:

`
{"additive",8,0,2,2,1691,0,0,,0,,0,,0,,[CHOOSE correction type"]}
"combined",8,0,2,2,1695},
"first_order",8,0,1,1,1687,0,0,,0,,0,,[CHOOSE correction order"]
"multiplicative",8,0,2,2,1693},
"second_order",8,0,1,1,1689},
"zeroth_order",8,0,1,1,1685}`

12.1.4.230 static KeyWord kw_227 () [static]
Initial value:

`
{"folds",9,0,1,0,1701,0,0,,0,,0,,[Number cross validation folds]
"percent",10,0,1,0,1703,0,0,,0,,0,,[Percent points per CV fold"}
}

12.1.4.231 static KeyWord kw_228 () [static]
Initial value:

`
{"cross_validate",8,2,1,0,1699,kw_227},
"press",8,0,2,0,1705,0,0,,0,,0,,[Perform PRESS cross validation]
"ModelCommands.html#ModelSurrG"}
}

12.1.4.232 static KeyWord kw_229 () [static]
Initial value:

`
{"annotated",8,0,1,0,1677},
"freeform",8,0,1,0,1679}
}

12.1.4.233 static KeyWord kw_230 () [static]
Initial value:

Dakota Version 5.4 Developers Manual generated on November 7, 2013
12.1.4.234  static KeyWord kw_231 ()  [static]

Initial value:

{  "point_selection",8,0,1,1,1567},
"trend",8,3,2,0,1569,kw_230,0.,0.,0.,0,"
{GP point selection} ModelCommands.html#ModelSurrG"
}

12.1.4.235  static KeyWord kw_232 ()  [static]

Initial value:

{  "constant",8,0,1,1,1581},
"linear",8,0,1,1,1583),
"quadratic",8,0,1,1,1587},
"reduced_quadratic",8,0,1,1,1585}

12.1.4.236  static KeyWord kw_233 ()  [static]

Initial value:

{  "correlation_lengths",14,0,5,0,1597,0,0.,0.,0.,0,"{Surfpack GP correlation lengths} ModelCommands.html#ModelSurrG"},
"export_model_file",11,0,6,0,1599),
"find_nugget",9,0,4,0,1595,0,0.,0.,0.,0,"{Surfpack finds the optimal nugget } ModelCommands.html#ModelSurrG"),
"max_trials",0x19,0,3,0,1591,0,0.,0.,0.,0,"{Surfpack GP maximum trials} ModelCommands.html#ModelSurrG"),
"nugget",0xa,0,4,0,1593,0,0.,0.,0.,0,"{Surfpack user-specified nugget } ModelCommands.html#ModelSurrG"),
"optimization_method",11,0,2,0,1589,0,0.,0.,0.,0,"{Surfpack GP optimization method} ModelCommands.html#ModelSurrG"),
"trend",8,4,1,0,1579,kw_232,0.,0.,0.,0,"{Surfpack GP trend function} ModelCommands.html#ModelSurrG"
}

12.1.4.237  static KeyWord kw_234 ()  [static]

Initial value:
12.1 Dakota Namespace Reference

12.1.4.238 static KeyWord kw_235 () [static]

Initial value:

```
[
    {"annotated",8,0,1,0,1671,0,,0,,0,,0,,0,"{Challenge file in annotated format} ModelCommands.html#ModelSurrG"},
    {"freeform",8,0,1,0,1673,0,,0,,0,,0,,0,"{Challenge file in freeform format} ModelCommands.html#ModelSurrG"}
]
```

12.1.4.239 static KeyWord kw_236 () [static]

Initial value:

```
[
    {"cubic",8,0,1,1,1609},
    {"linear",8,0,1,1,1607}
]
```

12.1.4.240 static KeyWord kw_237 () [static]

Initial value:

```
[
    {"export_model_file",11,0,3,0,1611},
    {"interpolation",8,2,2,0,1605,kw_236,0,,0,,0,,0,"{MARS interpolation} ModelCommands.html#ModelSurrG"},
    {"max_bases",9,0,1,0,1603,0,,0,,0,,0,,0,"{MARS maximum bases} ModelCommands.html#ModelSurrG"}
]
```

12.1.4.241 static KeyWord kw_238 () [static]

Initial value:

```
[
    {"export_model_file",11,0,3,0,1619},
    {"poly_order",9,0,1,0,1615,0,,0,,0,,0,"{MLS polynomial order} ModelCommands.html#ModelSurrG"},
    {"weight_function",9,0,2,0,1617,0,,0,,0,,0,"{MLS weight function} ModelCommands.html#ModelSurrG"}
]"
12.1.4.242 static KeyWord kw_239 () [static]

Initial value:

```
{
    "export_model_file",11,0,4,0,1629),
    "nodes",9,0,1,0,1623,0,0,0,0,0,"{ANN number nodes} ModelCommands.html#ModelSurrG"),
    "random_weight",9,0,3,0,1627,0,0,0,0,"{ANN random weight} ModelCommands.html#ModelSurrG"),
    "range",10,0,2,0,1625,0,0,0,0,"{ANN range} ModelCommands.html#ModelSurrG"
}
```

12.1.4.243 static KeyWord kw_240 () [static]

Initial value:

```
{
    "cubic",8,0,1,1,1649,0,0,0,0,"{CHOOSE polynomial order}"),
    "export_model_file",11,0,2,0,1651),
    "linear",8,0,1,1,1645),
    "quadratic",8,0,1,1,1647)
}
```

12.1.4.244 static KeyWord kw_241 () [static]

Initial value:

```
{
    "bases",9,0,1,0,1633,0,0,0,0,0,"{RBF number of bases} ModelCommands.html#ModelSurrG"),
    "export_model_file",11,0,5,0,1641),
    "max_pts",9,0,2,0,1635,0,0,0,0,"{RBF maximum points} ModelCommands.html#ModelSurrG"),
    "max_subsets",9,0,4,0,1639),
    "min_partition",9,0,3,0,1637,0,0,0,0,"{RBF minimum partition} ModelCommands.html#ModelSurrG"
}
```

12.1.4.245 static KeyWord kw_242 () [static]

Initial value:

```
{
    "all",8,0,1,1,1663),
    "none",8,0,1,1,1667),
    "region",8,0,1,1,1665)
}
```
12.1.4.246  static KeyWord kw_243 ()  [static]

Initial value:

```
{
  {"challenge_points_file",11,2,10,0,1707,kw_225,0.,0.,0.,0,"{Challenge file for surrogate metrics} ModelCommands.html#ModelSurrG"},
  {"correction",8,6,8,0,1683,kw_226,0.,0.,0.,0,"{Surrogate correction approach} ModelCommands.html#ModelSurrG"},
  {"dace_method_pointer",11,0,3,0,1659,0.,0.,0.,0,"{Design of experiments method pointer} ModelCommands.html#ModelSurrG"},
  {"diagnostics",7,2,9,0,1696,kw_228},
  {"export_points_file",11,2,6,0,1675,kw_229,0.,0.,0.,0,"{File export of global approximation-based sample results} ModelCommands.html#ModelSurrG"},
  {"gaussian_process",8,2,11,1563,kw_234,0.,0.,0.,0,"[CHOOSE surrogate type]{Dakota Gaussian process} ModelCommands.html#ModelSurrG"},
  {"import_points_file",11,2,5,0,1669,kw_235,0.,0.,0.,0,"{File import of samples for global approximation builds} ModelCommands.html#ModelSurrG"},
  {"kriging",0,2,1,1,1562,kw_234},
  {"mars",8,3,1,1,1601,kw_237,0.,0.,0.,0,"{Multivariate adaptive regression splines} ModelCommands.html#ModelSurrG"},
  {"metrics",15,2,9,0,1697,kw_228,0.,0.,0.,0,"{Compute surrogate diagnostics} ModelCommands.html#ModelSurrG"},
  {"minimum_points",8,0,2,0,1655},
  {"moving_least_squares",8,3,1,1,1613,kw_238,0.,0.,0.,0,"{Moving least squares} ModelCommands.html#ModelSurrG"},
  {"neural_network",8,4,1,1,1621,kw_239,0.,0.,0.,0,"{Artificial neural network} ModelCommands.html#ModelSurrG"},
  {"polynomial",8,4,1,1,1643,kw_240,0.,0.,0.,0,"{Polynomial} ModelCommands.html#ModelSurrG"},
  {"radial_basis",8,5,1,1,1631,kw_241},
  {"recommended_points",8,0,2,0,1657},
  {"reuse_points",8,3,4,0,1661,kw_242},
  {"reuse_samples",0,3,4,0,1660,kw_242},
  {"samples_file",3,2,5,0,1668,kw_235},
  {"total_points",9,0,2,0,1653},
  {"use_derivatives",8,0,7,0,1681,0.,0.,0.,0,"{Surfpack GP gradient enhancement} ModelCommands.html#ModelSurrG"}
}
```

12.1.4.247  static KeyWord kw_244 ()  [static]

Initial value:

```
{
  {"additive",8,0,2,2,1737,0.,0.,0.,0.,0,"[CHOOSE correction type]"},
  {"combined",8,0,2,2,1741},
  {"first_order",8,0,1,1,1733,0.,0.,0.,0,"[CHOOSE correction order]"},
  {"multiplicative",8,0,2,2,1739},
  {"second_order",8,0,1,1,1735},
  {"zeroth_order",8,0,1,1,1731}
}
```
12.1.4.248 static KeyWord kw_245 () [static]

Initial value:

```
{"correction",8,6,3,3,1729,kw_244,0.,0.,0.,0,"{Surrogate correction approach} ModelCommands.html#ModelSurrH"},
{"high_fidelity_model_pointer",11,0,2,2,1727,0.,0.,0.,0,"{Pointer to the high fidelity model specification} ModelCommands.html#ModelSurrH"},
{"low_fidelity_model_pointer",11,0,1,1,1725,0.,0.,0.,0,"{Pointer to the low fidelity model specification} ModelCommands.html#ModelSurrH"}
```

12.1.4.249 static KeyWord kw_246 () [static]

Initial value:

```
{"actual_model_pointer",11,0,2,2,1721,0.,0.,0.,0,"{Pointer to the truth model specification} ModelCommands.html#ModelSurrMP"},
{"taylor_series",8,0,1,1,1719,0.,0.,0.,0,"{Taylor series local approximation} ModelCommands.html#ModelSurrL"}
```

12.1.4.250 static KeyWord kw_247 () [static]

Initial value:

```
{"actual_model_pointer",11,0,2,2,1721,0.,0.,0.,0,"{Pointer to the truth model specification} ModelCommands.html#ModelSurrMP"},
{"tana",8,0,1,1,1715,0.,0.,0.,0,"{Two-point adaptive nonlinear approximation} ModelCommands.html#ModelSurrMP"}
```

12.1.4.251 static KeyWord kw_248 () [static]

Initial value:

```
{"global",8,21,2,1,1561,kw_243,0.,0.,0.,0,"{CHOOSE surrogate category}{Global approximations} ModelCommands.html#ModelSurrG"},
{"hierarchical",8,3,2,1,1723,kw_245,0.,0.,0.,0,"{Hierarchical approximation} ModelCommands.html#ModelSurrH"},
{"id_surrogates",13,0,1,0,1559,0.,0.,0.,0,"{Surrogate response ids} ModelCommands.html#ModelSurrogate"},
{"local",8,2,2,1,1717,kw_246,0.,0.,0.,0,"{Local approximation} ModelCommands.html#ModelSurrL"},
{"multipoint",8,2,2,1,1713,kw_247,0.,0.,0.,0,"{Multipoint approximation} ModelCommands.html#ModelSurrMP"}
```
12.1 Dakota Namespace Reference

12.1.4.252 static KeyWord kw_249 () [static]

Initial value:

```
{
    "hierarchical_tagging",8,0,4,0,1551,0,0,0,0,0,"{Hierarchical evaluation tags} ModelCommands.html#ModelIndControl"),
    "id_model",11,0,1,0,1545,0,0,0,0,0,"{Model set identifier} ModelCommands.html#ModelIndControl"),
    "nested",8,2,5,1,1743,kw_223,0,0,0,0,"{CHOOSE model type} ModelCommands.html#ModelIndControl"),
    "responses_pointer",11,0,3,0,1549,0,0,0,0,0,"{Responses set pointer} ModelCommands.html#ModelIndControl"),
    "single",8,1,5,1,1553,kw_224,0,0,0,0,"{Model set identifier} ModelCommands.html#ModelIndControl"),
    "surrogate",8,5,5,1,1557,kw_248),
    "variables_pointer",11,0,2,0,1547,0,0,0,0,0,"{Variables set pointer} ModelCommands.html#ModelIndControl")
}
```

12.1.4.253 static KeyWord kw_250 () [static]

Initial value:

```
{
    "annotated",8,0,3,0,2237,0,0,0,0,0,"{Data file in annotated format} RespCommands.html#RespFnLS"),
    "freeform",8,0,3,0,2239,0,0,0,0,0,"{Data file in freeform format} RespCommands.html#RespFnLS"),
    "num_config_variables",0x29,0,4,0,2241,0,0,0,0,"{Configuration variable columns in file} RespCommands.html#RespFnLS"),
    "num_experiments",0x29,0,1,0,2233,0,0,0,0,0,"{Experiments in file} RespCommands.html#RespFnLS"),
    "num_replicates",13,0,2,0,2235,0,0,0,0,0,"{Replicates per experiment in file} RespCommands.html#RespFnLS"),
    "num_std_deviations",0x29,0,5,0,2243,0,0,0,0,0,"{Standard deviation columns in file} RespCommands.html#RespFnLS")
}
```

12.1.4.254 static KeyWord kw_251 () [static]

Initial value:

```
{
    "nonlinear_equality_scale_types",0x807,0,2,0,2258,0,0,0,0,0,"{Nonlinear equality scale types} RespCommands.html#RespFnLS"),
    "nonlinear_equality_scales",0x806,0,3,0,2260,0,0,0,0,0,"{Nonlinear equality scales} RespCommands.html#RespFnLS"),
    "nonlinear_equality_targets",6,0,1,0,2256,0,0,0,0,0,"{Nonlinear equality targets} RespCommands.html#RespFnLS"),
    "scale_types",0x80f,0,2,0,2259,0,0,0,0,0,"{Scale types} RespCommands.html#RespFnLS"),
    "scales",0x80e,0,3,0,2261,0,0,0,0,0,"{Scales} RespCommands.html#RespFnLS"),
    "targets",14,0,1,0,2257,0,0,0,0,0,"{Targets} RespCommands.html#RespFnLS")
```
12.1.4.255 static KeyWord kw_252 () [static]

Initial value:

```
{"lower_bounds",14,0,1,0,2247,0,0,0,0,0,"{Nonlinear inequality lower bounds} RespCommands.html#RespFnLS",0,"nonlinear_inequality_constraints"},
{"nonlinear_inequality_lower_bounds",6,0,1,0,2246,0,0,0,0,0,"nonlinear_inequality_constraints"},
{"nonlinear_inequality_scale_types",0x807,0,3,0,2250,0,0,0,0,0,"nonlinear_inequality_constraints"},
{"nonlinear_inequality_scales",0x806,0,4,0,2252,0,0,0,0,0,"nonlinear_inequality_constraints"},
{"nonlinear_inequality_upper_bounds",6,0,2,0,2248,0,0,0,0,0,"nonlinear_inequality_constraints"},
{"scale_types",0x80f,0,3,0,2251,0,0,0,0,0,"nonlinear_inequality_constraints"},
{"scales",0x80e,0,4,0,2253,0,0,0,0,0,"nonlinear_inequality_constraints"},
{"upper_bounds",14,0,2,0,2249,0,0,0,0,0,"{Nonlinear inequality upper bounds} RespCommands.html#RespFnLS",0,"nonlinear_inequality_constraints"}
```

12.1.4.256 static KeyWord kw_253 () [static]

Initial value:

```
{"calibration_data_file",11,6,4,0,2231,kw_250,0,0,0,0,"{Calibration data file name} RespCommands.html#RespFnLS"},
{"calibration_term_scale_types",0x807,0,1,0,2224,0,0,0,0,0,"calibration_terms"},
{"calibration_term_scales",0x806,0,2,0,2226,0,0,0,0,0,"calibration_terms"},
{"calibration_weights",6,0,3,0,2228,0,0,0,0,0,"calibration_terms"},
{"least_squares_data_file",3,6,4,0,2230,kw_250},
{"least_squares_term_scale_types",0x807,0,1,0,2224,0,0,0,0,0,"calibration_terms"},
{"least_squares_term_scales",0x806,0,2,0,2226,0,0,0,0,0,"calibration_terms"},
{"least_squares_weights",6,0,3,0,2228,0,0,0,0,0,"calibration_terms"},
{"nonlinear_inequality_constraints",0x29,6,6,0,2255,kw_251,0,0,0,0,"{Number of nonlinear equality constraints} RespCommands.html#RespFnLS"},
{"nonlinear_inequality_constraints",0x29,8,5,0,2245,kw_252,0,0,0,0,"{Number of nonlinear equality constraints} RespCommands.html#RespFnLS"},
{"num_nonlinear_inequality_constraints",0x21,6,6,0,2254,kw_251},
{"num_nonlinear_inequality_constraints",0x21,8,5,0,2244,kw_252},
{"primary_scale_types",0x80f,0,1,0,2225,0,0,0,0,0,"{Calibration scaling types} RespCommands.html#RespFnLS"},
{"primary_scales",0x80e,0,2,0,2227,0,0,0,0,0,"{Calibration scales} RespCommands.html#RespFnLS"},
{"weights",14,0,3,0,2229,0,0,0,0,0,"{Calibration term weights} RespCommands.html#RespFnLS"}
```
12.1.4.257  static KeyWord kw_254 ()  [static]

Initial value:

```
{"absolute",8,0,2,0,2285},
{"bounds",8,0,2,0,2287},
{"ignore_bounds",8,0,1,0,2281,0,0.,0.,0.0,"{Ignore variable bounds} RespCommands.html#RespGradMixed"},
{"relative",8,0,2,0,2283}
```

12.1.4.258  static KeyWord kw_255 ()  [static]

Initial value:

```
{"central",8,0,6,0,2295,0,0.,0.,0.0,"{CHOOSE difference interval} RespCommands.html#RespGradNum"},
{"dakota",8,4,4,0,2279,kw_254,0.,0.,0.,0.,0,"{CHOOSE gradient source} RespCommands.html#RespGradNum"},
{"fd_gradient_step_size",0x406,0,7,0,2296,0,0.,0.,0.001},
{"fd_step_size",0x40e,0,7,0,2297,0,0.,0.,0.001,0,"{Finite difference step size} RespCommands.html#RespGradMixed"},
{"forward",8,0,6,0,2293,0,0.,0.,0.0,"{Method source} RespCommands.html#RespGradNum"},
{"id_analytic_gradients",13,0,2,2,2273,0,0.,0.,0.0,"{Analytic derivatives function list} RespCommands.html#RespGradMixed"},
{"id_numerical_gradients",13,0,2,1,2271,0,0.,0.,0.0,"{Numerical derivatives function list} RespCommands.html#RespGradMixed"},
{"interval_type",8,0,5,0,2291,0,0.,0.,0.,0,"{Interval type} RespCommands.html#RespGradNum"},
{"method_source",8,0,3,0,2277,0,0.,0.,0.0,"{Method source} RespCommands.html#RespGradNum"},
{"vendor",8,0,4,0,2289}
```

12.1.4.259  static KeyWord kw_256 ()  [static]

Initial value:

```
{"fd_hessian_step_size",6,0,1,0,2328},
{"fd_step_size",14,0,1,0,2329,0,0.,0.,0.0,"{Finite difference step size} RespCommands.html#RespHessMixed"}
```

12.1.4.260  static KeyWord kw_257 ()  [static]

Initial value:

```
{"damped",8,0,1,0,2345,0,0.,0.,0.,0.0,"{Numerical safeguarding of BFGS update} RespCommands.html#RespHessMixed"}
```
12.1.4.261 static KeyWord kw_258 () [static]

Initial value:

```cpp
{"bfgs",8,1,1,2343,kw_257,0.,0.,0.,0,"[CHOOSE Hessian approx.]"},
{"sr1",8,0,1,2347}
```

12.1.4.262 static KeyWord kw_259 () [static]

Initial value:

```cpp
{"absolute",8,0,2,0,2333},
{"bounds",8,0,2,0,2335},
{"central",8,0,3,0,2339,0.,0.,0.,0,"[CHOOSE difference interval]"},
{"forward",8,0,3,0,2337,0.,0.,0.,0,"@"},
{"id_analytic_hessians",13,0,5,0,2349,0.,0.,0.,0,"{Analytic Hessians function list} RespCommands.html#RespHessMixed"},
{"id_numerical_hessians",13,2,4,0,2341,kw_258,0.,0.,0.,0,"{Quasi Hessians function list} RespCommands.html#RespHessMixed"},
{"relative",8,0,2,0,2331}
```

12.1.4.263 static KeyWord kw_260 () [static]

Initial value:

```cpp
{"nonlinear_equality_scale_types",0x807,0,2,0,2218,0.,0.,0.,0,0,0,"nonlinear_equality_constraints"},
{"nonlinear_equality_scales",0x806,0,3,0,2220,0.,0.,0.,0,0,0,"nonlinear_equality_constraints"},
{"nonlinear_equality_targets",6,0,1,0,2216,0.,0.,0.,0,0,0,"nonlinear_equality_constraints"},
{"scale_types",0x805,0,2,0,2219,0.,0.,0.,0,0,0,"Nonlinear scaling types (for inequalities or equalities) RespCommands.html#RespFnLS",0,"nonlinear_equality_constraints"},
{"scales",0x804,0,3,0,2221,0.,0.,0.,0,0,0,"Nonlinear scales (for inequalities or equalities) RespCommands.html#RespFnLS",0,"nonlinear_equality_constraints"},
{"targets",14,0,1,0,2217,0.,0.,0.,0,0,0,"Nonlinear equality constraint targets RespCommands.html#RespFnOpt",0,"nonlinear_equality_constraints"}
```

12.1.4.264 static KeyWord kw_261 () [static]

Initial value:
12.1 Dakota Namespace Reference

12.1.4.265 static KeyWord kw_262 () [static]

Initial value:

```c++
{
    "lower_bounds",14,0,1,0,2207,0,0.,0.,0.,0.,0,"{Nonlinear inequality
    constraint lower bounds} RespCommands.html#RespFnOpt",0,"nonlinear_inequality_con
    straints"),
    "nonlinear_inequality_lower_bounds",6,0,1,0,2206,0,0.,0.,0.,0,0,"{Nonlinear inequality
    constraint lower bounds} RespCommands.html#RespFnOpt",0,"nonlinear_inequality_co
    nstraints"),
    "nonlinear_inequality_scale_types",0x807,0,3,0,2210,0,0.,0.,0.,0,0,"{Nonlinear inequality
    constraints}
    "nonlinear_inequality_scales",0x806,0,4,0,2212,0,0.,0.,0.,0,0,"{Nonlinear inequality
    constraints}
    "nonlinear_inequality_upper_bounds",6,0,2,0,2208,0,0.,0.,0.,0,0,"{Nonlinear inequality
    constraint upper bounds} RespCommands.html#RespFnOpt",0,"nonlinear_inequality_co
    nstraints"
    "scale_types",0x80f,0,3,0,2211,0,0.,0.,0.,0,0,"{Nonlinear constraint scaling types (for inequalities or equalities)} RespCommands.html#RespFnOpt",0,"nonlinear_inequality_constraints"
    "scales",0x80e,0,4,0,2213,0,0.,0.,0.,0,"{Nonlinear constraint scales (for inequalities or equalities)} RespCommands.html#RespFnOpt",0,"nonlinear_inequality_constraints"
    "upper_bounds",14,0,2,0,2209,0,0.,0.,0.,0,"{Nonlinear inequality
    constraint upper bounds} RespCommands.html#RespFnOpt",0,"nonlinear_inequality_co
    nstraints"
}
```

12.1.4.266 static KeyWord kw_263 () [static]

Initial value:

```c++
{
    "multi_objective_weights",6,0,4,0,2202,0,0.,0.,0.,0,0,"{Objective
    functions}
    "nonlinear_equality_constraints",0x29,6,6,0,2215,kw_260,0,0.,0.
    ,0,"{Number of nonlinear equality constraints} RespCommands.html#RespFnOpt",0,"nonlinear_equality_constraints"
    "num_nonlinear_equality_constraints",0x21,6,6,0,2214,kw_260
    "num_nonlinear_inequality_constraints",0x21,8,5,0,2205,kw_261
    "objective_function_scale_types",0x807,0,2,0,2198,0,0.,0.
    ,0,0,"{Objective function scaling types} RespCommands.html#RespFnOpt",0,"objective_functions"
    "objective_function_scales",0x806,0,3,0,2200,0,0.,0.,0.,0,0,"{Objective
    functions}
    "primary_scale_types",0x80f,0,2,0,2199,0,0.,0.,0.,0,"{Objective function scaling types} RespCommands.html#RespFnOpt",0,"objective_functions"
    "primary_scales",0x80e,0,3,0,2201,0,0.,0.,0.,0,"{Objective function scales} RespCommands.html#RespFnOpt",0,"objective_functions"
    "sense",0x80f,0,1,0,2197,0,0.,0.,0.,0,"{Optimization sense} Resp Commands.html#RespFnOpt",0,"objective_functions"
    "weights",14,0,4,0,2203,0,0.,0.,0.,0,"{Multi-objective weighting
    s} RespCommands.html#RespFnOpt",0,"objective_functions"
}
```
12.1.4.267 static KeyWord kw_264 () [static]

Initial value:

```cpp
{
    "absolute", 8, 0, 2, 0, 2307,
    "bounds", 8, 0, 2, 0, 2309,
    "central", 8, 0, 3, 0, 2313, 0, 0, 0, 0, 0, 0, 0, 0,
    "forward", 8, 0, 6, 0, 2293, 0, 0, 0, 0, 0, 0,
    "interval", 8, 0, 5, 0, 2291, 0, 0, 0, 0, 0, 0,
    "relative", 8, 0, 2, 0, 2305
}
```

12.1.4.268 static KeyWord kw_265 () [static]

Initial value:

```cpp
{
    "damped", 8, 0, 1, 0, 2319, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    "forward", 8, 0, 3, 0, 2311, 0, 0, 0, 0, 0, 0,
    "relative", 8, 0, 2, 0, 2305
}
```

12.1.4.269 static KeyWord kw_266 () [static]

Initial value:

```cpp
{
    "bfgs", 8, 1, 1, 1, 2317, kw_265, 0, 0, 0, 0, 0, 0, 0, 0,
    "sr1", 8, 0, 1, 1, 2321
}
```

12.1.4.270 static KeyWord kw_267 () [static]

Initial value:
12.1 Dakota Namespace Reference

12.1.4.271 static KeyWord kw_268 () [static]

Initial value:

```cpp
{ "method_list",15,0,1,1,39,0,0,0,0,0,"{List of methods} StratCommands.html#StratHybrid" }
```

12.1.4.272 static KeyWord kw_269 () [static]

Initial value:

```cpp
{ "global_method_pointer",11,0,1,1,31,0,0,0,0,0,"{Pointer to the global method specification} StratCommands.html#StratHybrid" },
{ "local_method_pointer",11,0,2,2,33,0,0,0,0,0,"{Pointer to the local method specification} StratCommands.html#StratHybrid" },
{ "local_search_probability",10,0,3,0,35,0,0,0,0,0,"{Probability of executing local searches} StratCommands.html#StratHybrid" }
```
12.1.4.273 static KeyWord kw_270 () [static]

Initial value:

```
{"method_list",15,0,1,1,27,0,0.,0.,0.,0,"{List of methods} StratCommands.html#StratHybrid"}
```

12.1.4.274 static KeyWord kw_271 () [static]

Initial value:

```
{"collaborative",8,1,1,37,kw_268,0.,0.,0.,0,"[CHOOSE hybrid type] Collaborative hybrid} StratCommands.html#StratHybrid"},
{"coupled",0,3,1,1,28,kw_269},
{"embedded",8,3,1,1,29,kw_269,0.,0.,0.,0,"[Embedded hybrid] StratCommands.html#StratHybrid"},
{"sequential",8,1,1,25,kw_270,0.,0.,0.,0,"[Sequential hybrid] StratCommands.html#StratHybrid"},
{"uncoupled",0,1,1,1,24,kw_270}
```

12.1.4.275 static KeyWord kw_272 () [static]

Initial value:

```
{"master",8,0,1,1,19},
{"peer",8,0,1,1,21}
```

12.1.4.276 static KeyWord kw_273 () [static]

Initial value:

```
{"seed",9,0,1,0,47,0.,0.,0.,0,"{Seed for random starting points} StratCommands.html#StratMultiStart"}
```

12.1.4.277 static KeyWord kw_274 () [static]

Initial value:

```
{"method_pointer",11,0,1,1,43,0.,0.,0.,0,"{Method pointer} StratCommands.html#StratMultiStart"},
{"random_starts",9,1,2,0,45,kw_273,0.,0.,0.,0,"{Number of random
12.1.4.278 static KeyWord kw_275 () [static]

Initial value:

{  
  {"seed",9,0,1,0,57,0,0,.0,.0,.0,"{Seed for random weighting sets} StratCommands.html#StratParetoSet"},
  {"starting_points",14,0,3,0,49,0,0,.0,.0,.0,"{List of user-specified starting points} StratCommands.html#StratMultiStart"},
}

12.1.4.279 static KeyWord kw_276 () [static]

Initial value:

{  
  {"method_pointer",11,0,1,1,53,0,0,.0,.0,.0,"{Optimization method pointer} StratCommands.html#StratParetoSet"},
  {"multi_objective_weight_sets",6,0,3,0,58},
  {"opt_method_pointer",3,0,1,1,52},
  {"random_weight_sets",9,1,2,0,55,kw_275,0,.0,.0,.0,"{Number of random weighting sets} StratCommands.html#StratParetoSet"},
  {"weight_sets",14,0,3,0,59,0,.0,.0,.0,.0,"{List of user-specified weighting sets} StratCommands.html#StratParetoSet"},
}

12.1.4.280 static KeyWord kw_277 () [static]

Initial value:

{  
  {"results_output_file",11,0,1,0,13,0,0,.0,.0,.0,"{File name for results output} StratCommands.html#StratIndControl"},
}

12.1.4.281 static KeyWord kw_278 () [static]

Initial value:

{  
  {"method_pointer",11,0,1,0,63,0,.0,.0,.0,.0,"{Method pointer} StratCommands.html#StratSingle"},
}
12.1.4.282 static KeyWord kw_279 () [static]

Initial value:

```c
{
  "tabular_graphics_file",11,0,1,0,7,0,0,,0,,0,,0,"{File name for tabular graphics data} StratCommands.html#StratIndControl"}
```

12.1.4.283 static KeyWord kw_280 () [static]

Initial value:

```c
{
  "graphics",8,0,1,0,3,0,0,,0,,0,,0,"{Graphics flag} StratCommands.html#StratIndControl"),
  "hybrid",8,5,7,1,23,kw_271,0,,0,,0,,0,"{CHOOSE strategy type} Hybrid strategy} StratCommands.html#StratHybrid"),
  "iterator_scheduling",8,2,6,0,17,kw_272,0,,0,,0,,0,"{Message passing configuration for scheduling of iterator jobs} StratCommands.html#StratIndControl"),
  "iterator_servers",9,0,5,0,15,0,0,,0,,0,,0,"{Number of iterator servers} StratCommands.html#StratIndControl"),
  "multi_start",8,3,7,1,41,kw_274,0,,0,,0,,0,"{Multi-start iteration strategy} StratCommands.html#StratMultiStart"),
  "output_precision",0x29,0,3,0,9,0,0,,0,,0,,0,"{Numeric output precision} StratCommands.html#StratIndControl"),
  "pareto_set",8,5,7,1,51,kw_276,0,,0,,0,,0,"{Pareto set optimization strategy} StratCommands.html#StratParetoSet"),
  "results_output",8,1,4,0,11,kw_277,0,,0,,0,,0,"{Enable results output} StratCommands.html#StratIndControl"),
  "single_method",8,1,7,1,61,kw_278,0,,0,,0,,0,"{Single method strategy} StratCommands.html#StratSingle"),
  "tabular_graphics_data",8,1,2,0,5,kw_279,0,,0,,0,,0,"{Tabulation of graphics data} StratCommands.html#StratIndControl")
```

12.1.4.284 static KeyWord kw_281 () [static]

Initial value:

```c
{
  "aleatory",8,0,1,1,1771),
  "all",8,0,1,1,1765),
  "design",8,0,1,1,1767),
  "epistemic",8,0,1,1,1773),
  "state",8,0,1,1,1775),
  "uncertain",8,0,1,1,1769)
```

12.1.4.285 static KeyWord kw_282 () [static]

Initial value:
12.1 Dakota Namespace Reference

```csharp
{
    {"alphas",14,0,1,1,1889,0,0.,0.,0.,0.,"{beta uncertain alphas} VarCommands.html#VarCAUV_Beta",0,"beta_uncertain"},
    {"betas",14,0,2,2,1891,0,0.,0.,0.,0.,"{beta uncertain betas} VarCommands.html#VarCAUV_Beta",0,"beta_uncertain"},
    {"buv_alphas",6,0,1,1,1888,0,0.,0.,0.,0.,0,0,0,"beta_uncertain"},
    {"buv_betas",6,0,2,2,1890,0,0.,0.,0.,0.,0,0,0,"beta_uncertain"},
    {"buv_descriptors",7,0,5,0,1896,0,0.,0.,0.,0,0,0,0,"beta_uncertain"},
    {"buv_lower_bounds",6,0,3,3,1892,0,0.,0.,0.,0,0,0,0,"beta_uncertain"},
    {"buv_upper_bounds",6,0,4,4,1894,0,0.,0.,0.,0,0,0,0,"beta_uncertain"},
    {"descriptors",15,0,5,0,1897,0,0.,0.,0.,0,"{Descriptors} VarCommands.html#VarCAUV_Beta",0,"beta_uncertain"},
    {"lower_bounds",14,0,3,3,1893,0,0.,0.,0.,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Beta",0,"beta_uncertain"},
    {"upper_bounds",14,0,4,4,1895,0,0.,0.,0.,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Beta",0,"beta_uncertain"}
}
```

12.1.4.286 static KeyWord kw_283 () [static]

Initial value:

```csharp
{ "descriptors",15,0,3,0,1955,0,0.,0.,0.,0,"{Descriptors} VarCommands.html#VarDAUV_Binomial",0,"binomial_uncertain"},
{ "num_trials",13,0,2,2,1953,0,0.,0.,0.,0,"{binomial uncertain num_trials} VarCommands.html#VarDAUV_Binomial",0,"binomial_uncertain"},
{ "prob_per_trial",6,0,1,1,1950,0,0.,0.,0.,0,"{binomial uncertain probability_per_trial} VarCommands.html#VarDAUV_Binomial",0,"binomial_uncertain"}
```

12.1.4.287 static KeyWord kw_284 () [static]

Initial value:

```csharp
{ "cdv_descriptors",7,0,6,0,1792,0,0.,0.,0.,0,0,0,0,"continuous_design"},
{ "cdv_initial_point",6,0,1,0,1782,0,0.,0.,0.,0,0,0,0,"continuous_design"},
{ "cdv_lower_bounds",6,0,2,0,1784,0,0.,0.,0.,0,0,0,0,"continuous_design"},
{ "cdv_scale_types",0x807,0,4,0,1788,0,0.,0.,0.,0,0,0,0,"continuous_design"},
{ "cdv_scales",0x806,0,5,0,1790,0,0.,0.,0.,0,0,0,0,"continuous_design"},
{ "cdv_upper_bounds",6,0,3,0,1786,0,0.,0.,0.,0,0,0,0,"continuous_design"},
{ "descriptors",15,0,6,0,1793,0,0.,0.,0.,0,0,0,0,"{Descriptors} VarCommands.html#VarCDV",0,"continuous_design"},
{ "initial_point",14,0,1,0,1783,0,0.,0.,0.,0,0,0,0,"{Initial point} VarCommands.html#VarCDV",0,"continuous_design"},
```

Dakota Version 5.4 Developers Manual generated on November 7, 2013
220 Namespace Documentation

{{"lower_bounds",14,0,2,0,1785,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarCDV",0,"continuous_design"},
{"scale_types",0x80f,0,4,0,1789,0,0,0,0,0,"{Scaling types} VarCommands.html#VarCDV",0,"continuous_design"},
{"scales",0x80e,0,5,0,1791,0,0,0,0,0,"{Scales} VarCommands.html#VarCDV",0,"continuous_design"},
{"upper_bounds",14,0,3,0,1787,0,0,0,0,0,"{Upper bounds} VarCommands.html#VarCDV",0,"continuous_design"}}

12.1.4.288 static KeyWord kw_285 () [static]

Initial value:

{{"descriptors",15,0,5,0,2003,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCEUV_Interval",0,"continuous_interval_uncertain"},
{"interval_probabilities",14,0,2,0,1997,0,0,0,0,0,"{Basic probability assignments per continuous interval} VarCommands.html#VarCEUV_Interval"},
{"interval_probs",6,0,2,0,1996},
{"iuv_descriptors",7,0,5,0,2002,0,0,0,0,0,0,0,"{Continuous interval uncertain}"},
{"iuv_interval_probs",6,0,2,0,1996},
{"iuv_num_intervals",5,0,1,0,1994,0,0,0,0,0,0,0,"{Number of intervals defined for each continuous interval variable} VarCommands.html#VarCEUV_Interval"},
{"lower_bounds",14,0,3,1,1999,0,0,0,0,0,"{Lower bounds of continuous intervals} VarCommands.html#VarCEUV_Interval"},
{"num_intervals",13,0,1,0,1995,0,0,0,0,0,0,0,"{Number of intervals defined for each continuous interval variable} VarCommands.html#VarCEUV_Interval"},
{"upper_bounds",14,0,4,2,2001,0,0,0,0,0,"{Upper bounds of continuous intervals} VarCommands.html#VarCEUV_Interval"}}

12.1.4.289 static KeyWord kw_286 () [static]

Initial value:

{{"csv_descriptors",7,0,4,0,2044,0,0,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCSV",0,"continuous_state"},
{"csv_initial_state",6,0,1,0,2038,0,0,0,0,0,0,0,0,"{Continuous state}"},
{"csv_lower_bounds",6,0,2,0,2040,0,0,0,0,0,0,0,0,"{Continuous state}"},
{"csv_upper_bounds",6,0,3,0,2042,0,0,0,0,0,0,0,0,"{Continuous state}"},
{"descriptors",15,0,4,0,2045,0,0,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCSV",0,"continuous_state"},
{"initial_state",14,0,1,0,2039,0,0,0,0,0,0,0,0,"{Initial state} VarCommands.html#VarCSV",0,"continuous_state"},
{"lower_bounds",14,0,2,0,2041,0,0,0,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarCSV",0,"continuous_state"},
{"upper_bounds",14,0,3,0,2043,0,0,0,0,0,0,0,0,"{Upper bounds} VarCommands.html#VarCSV",0,"continuous_state"}}
12.1.4.290 static KeyWord kw_287 () [static]

Initial value:

```{cpp}
{
    {"ddv_descriptors",7,0,4,0,1802,0,0,0,0,0,0,0,"discrete_design_range"},
    {"ddv_initial_point",5,0,1,0,1796,0,0,0,0,0,0,0,"discrete_design_range"},
    {"ddv_lower_bounds",5,0,2,0,1798,0,0,0,0,0,0,0,"discrete_design_range"},
    {"ddv_upper_bounds",5,0,3,0,1800,0,0,0,0,0,0,0,"discrete_design_range"},
    {"descriptors",15,0,4,0,1803,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDDRIV"},
    {"initial_point",13,0,1,0,1797,0,0,0,0,0,0,0,"{Initial point} VarCommands.html#VarDDRIV"},
    {"lower_bounds",13,0,2,0,1799,0,0,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarDDRIV"},
    {"upper_bounds",13,0,3,0,1801,0,0,0,0,0,0,0,"{Upper bounds} VarCommands.html#VarDDRIV"}
}
```

12.1.4.291 static KeyWord kw_288 () [static]

Initial value:

```{cpp}
{
    {"descriptors",15,0,4,0,1813,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDDSIV"},
    {"initial_point",13,0,1,0,1807,0,0,0,0,0,0,0,"{Initial point} VarCommands.html#VarDDSIV"},
    {"num_set_values",13,0,2,0,1809,0,0,0,0,0,0,0,"{Number of values for each variable} VarCommands.html#VarDDSIV"},
    {"set_values",13,0,3,1,1811,0,0,0,0,0,0,0,"{Set values} VarCommands.html#VarDDSIV"}
}
```

12.1.4.292 static KeyWord kw_289 () [static]

Initial value:

```{cpp}
{
    {"descriptors",15,0,4,0,1823,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDDSRV"},
    {"initial_point",14,0,1,0,1817,0,0,0,0,0,0,0,"{Initial point} VarCommands.html#VarDDSRV"},
    {"num_set_values",13,0,2,0,1819,0,0,0,0,0,0,0,"{Number of values for each variable} VarCommands.html#VarDDSRV"},
    {"set_values",14,0,3,1,1821,0,0,0,0,0,0,0,"{Set values} VarCommands.html#VarDDSRV"}
}
```
12.1.4.293  static KeyWord kw_290 ()  [static]

Initial value:

```
{
    "{descriptors",15,0,5,0,2015,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDIUV",0,"discrete_interval_uncertain"},
    "{interval_probabilities",14,0,2,0,2009,0,0,0,0,0,"{Basic probability assignments per interval} VarCommands.html#VarDIUV"},
    "{interval_probs",6,0,2,0,2008},
    "{lower_bounds",13,0,3,1,2011,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarDIUV"},
    "{num_intervals",13,0,1,0,2007,0,0,0,0,0,"{Number of intervals defined for each interval variable} VarCommands.html#VarDIUV",0,"discrete_interval_uncertain"},
    "{range_probabilities",6,0,2,0,2008},
    "{range_probs",6,0,2,0,2008},
    "{upper_bounds",13,0,4,2,2013,0,0,0,0,0,"{Upper bounds} VarCommands.html#VarDIUV"},
}
```

12.1.4.294  static KeyWord kw_291 ()  [static]

Initial value:

```
{
    "{descriptors",15,0,4,0,2055,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDSRIV",0,"discrete_state_range"},
    "{dsv_descriptors",7,0,4,0,2054,0,0,0,0,0,0,0,"discrete_state_range"},
    "{dsv_initial_state",5,0,1,0,2048,0,0,0,0,0,0,0,"discrete_state_range"},
    "{dsv_lower_bounds",5,0,2,0,2050,0,0,0,0,0,0,0,"discrete_state_range"},
    "{dsv_upper_bounds",5,0,3,0,2052,0,0,0,0,0,0,0,"discrete_state_range"},
    "{initial_state",13,0,1,0,2049,0,0,0,0,0,0,0,"{Initial states} VarCommands.html#VarDSRIV",0,"discrete_state_range"},
    "{lower_bounds",13,0,2,0,2051,0,0,0,0,0,0,0,"{Lower bounds} VarCommands.html#VarDSRIV",0,"discrete_state_range"},
    "{upper_bounds",13,0,3,0,2053,0,0,0,0,0,0,0,"{Upper bounds} VarCommands.html#VarDSRIV",0,"discrete_state_range"},
}
```

12.1.4.295  static KeyWord kw_292 ()  [static]

Initial value:

```
{
    "{descriptors",15,0,4,0,2065,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDSSIV",0,"discrete_state_set_integer"},
    "{initial_state",13,0,1,0,2059,0,0,0,0,0,"{Initial state} VarCommands.html#VarDSSIV",0,"discrete_state_set_integer"},
    "{num_set_values",13,0,2,0,2061,0,0,0,0,0,"{Number of values for each variable} VarCommands.html#VarDSSIV",0,"discrete_state_set_integer"},
    "{set_values",13,0,3,1,2063,0,0,0,0,0,"{Set values} VarCommands.html#VarDSSIV",0,"discrete_state_set_integer"},
}
```
12.1.4.296 static KeyWord kw_293 () [static]

Initial value:

```
{
    "descriptors", 15, 0, 4, 0, 2075, 0, 0, 0, 0, 0, 0, "{Descriptors} VarCommands.html#VarDSSIV", 0, "discrete_state_set_real"},
    "initial_state", 14, 0, 1, 0, 2069, 0, 0, 0, 0, 0, "{Initial state} VarCommands.html#VarDSSIV", 0, "discrete_state_set_real"},
    "num_set_values", 13, 0, 2, 0, 2071, 0, 0, 0, 0, 0, "{Number of values for each variable} VarCommands.html#VarDSSIV", 0, "discrete_state_set_real"},
    "set_values", 14, 0, 3, 1, 2073, 0, 0, 0, 0, 0, "{Set values} VarCommands.html#VarDSSIV"}
```

12.1.4.297 static KeyWord kw_294 () [static]

Initial value:

```
{
    "descriptors", 15, 0, 4, 0, 2025, 0, 0, 0, 0, 0, 0, "{Descriptors} VarCommands.html#VarDUSIV", 0, "discrete_uncertain_set_integer"},
    "num_set_values", 13, 0, 1, 0, 2019, 0, 0, 0, 0, 0, "{Number of values for each variable} VarCommands.html#VarDUSIV", 0, "discrete_uncertain_set_integer"},
    "set_probabilities", 14, 0, 3, 0, 2023, 0, 0, 0, 0, 0, "{Probabilities for each set member} VarCommands.html#VarDUSIV"},
    "set_probs", 6, 0, 3, 0, 2022},
    "set_values", 13, 0, 2, 1, 2021, 0, 0, 0, 0, 0, "{Set values} VarCommands.html#VarDUSIV"}
```

12.1.4.298 static KeyWord kw_295 () [static]

Initial value:

```
{
    "descriptors", 15, 0, 4, 0, 2035, 0, 0, 0, 0, 0, 0, "{Descriptors} VarCommands.html#VarDUSRV", 0, "discrete_uncertain_set_real"},
    "num_set_values", 13, 0, 1, 0, 2029, 0, 0, 0, 0, 0, "{Number of values for each variable} VarCommands.html#VarDUSRV", 0, "discrete_uncertain_set_real"},
    "set_probabilities", 14, 0, 3, 0, 2033, 0, 0, 0, 0, 0, "{Probabilities for each set member} VarCommands.html#VarDUSRV"},
    "set_probs", 6, 0, 3, 0, 2032},
    "set_values", 14, 0, 2, 1, 2031, 0, 0, 0, 0, 0, "{Set values} VarCommands.html#VarDUSRV"}
```
12.1.4.299  static KeyWord kw_296 ()  [static]

Initial value:

```
{
   {"betas",14,0,1,1,1883,0,0.,0.,0.,0.,0,"{exponential uncertain betas
   } VarCommands.html#VarCAUV_Exponential",0,"exponential_uncertain"},
   {"descriptors",15,0,2,0,1885,0,0.,0.,0.,0,"{Descriptors} VarComma
   nds.html#VarCAUV_Exponential",0,"exponential_uncertain"},
   {"euv_betas",6,0,1,1,1882,0,0.,0.,0.,0,0,0,"exponential_uncer
   tain"},
   {"euv_descriptors",7,0,2,0,1884,0,0.,0.,0,0,0,0,"exponential_uncer
   tain"}
}
```

12.1.4.300  static KeyWord kw_297 ()  [static]

Initial value:

```
{
   {"alphas",14,0,1,1,1917,0,0.,0.,0.,0,"{frechet uncertain alphas} Va
   rCommands.html#VarCAUV_Frechet",0,"frechet_uncertain"},
   {"betas",14,0,2,2,1919,0,0.,0.,0.,0,0,"{frechet uncertain betas} Va
   rCommands.html#VarCAUV_Frechet",0,"frechet_uncertain"},
   {"descriptors",15,0,3,0,1921,0,0.,0.,0.,0,0,0,"{Descriptors} VarComma
   nds.html#VarCAUV_Frechet",0,"frechet_uncertain"},
   {"fuv_alphas",6,0,1,1,1916,0,0.,0.,0,0,0,0,"frechet_uncertain"},
   {"fuv_betas",6,0,2,2,1918,0,0.,0.,0,0,0,0,"frechet_uncertain"},
   {"fuv_descriptors",7,0,3,0,1920,0,0.,0.,0,0,0,0,"frechet_uncertainty"}
}
```

12.1.4.301  static KeyWord kw_298 ()  [static]

Initial value:

```
{
   {"alphas",14,0,1,1,1901,0,0.,0.,0.,0,"{gamma uncertain alphas} Va
   rCommands.html#VarCAUV_Gamma",0,"gamma_uncertain"},
   {"betas",14,0,2,2,1903,0,0.,0.,0.,0,0,"{gamma uncertain betas} VarC
   ommands.html#VarCAUV_Gamma",0,"gamma_uncertain"},
   {"descriptors",15,0,3,0,1905,0,0.,0.,0.,0,0,0,"{Descriptors} VarComma
   nds.html#VarCAUV_Gamma",0,"gamma_uncertain"},
   {"gauv_alphas",6,0,1,1,1900,0,0.,0.,0,0,0,0,"gamma_uncertain"},
   {"gauv_betas",6,0,2,2,1902,0,0.,0.,0,0,0,0,"gamma_uncertain"},
   {"gauv_descriptors",7,0,3,0,1904,0,0.,0.,0,0,0,0,"gamma_uncertain"}
}
```

12.1.4.302  static KeyWord kw_299 ()  [static]

Initial value:
12.1 Dakota Namespace Reference

12.1.4.303 static KeyWord kw_300 () [static]

Initial value:

{  
  {"alphas",14,0,1,1,1909,0,0,0,0,0,0,"{gumbel uncertain alphas} VarCommands.html#VarCAUV_Gumbel"},  
  {"betas",14,0,2,2,1911,0,0,0,0,0,0,"{gumbel uncertain betas} VarCommands.html#VarCAUV_Gumbel"},  
  {"descriptors",15,0,3,0,1913,0,0,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDAUV_Geometric"},  
  {"probability_per_trial",14,0,1,1,1967,0,0,0,0,0,0,0,0,"geometric_uncertain"}  
}

12.1.4.304 static KeyWord kw_301 () [static]

Initial value:

{  
  {"abscissas",14,0,2,1,1935,0,0,0,0,0,0,"{sets of abscissas for bin-based histogram variables} VarCommands.html#VarCAUV_Bin_Histogram"},  
  {"counts",14,0,3,2,1939,0,0,0,0,0,0,"{sets of counts for bin-based histogram variables} VarCommands.html#VarCAUV_Bin_Histogram"},  
  {"descriptors",15,0,4,0,1941,0,0,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV_Bin_Histogram"},  
  {"huv_bin_abscissas",6,0,1,1,1908,0,0,0,0,0,0,0,0,"gumbel_uncertain"},  
  {"huv_bin_counts",6,0,3,2,1938},  
  {"huv_bin_descriptors",7,0,3,0,1912,0,0,0,0,0,0,0,0,"histogram_bin_uncertain"},  
  {"huv_bin_ordinates",6,0,3,2,1936},  
  {"huv_num_bin_pairs",5,0,1,0,1932,0,0,0,0,0,0,0,0,"histogram_bin_uncertain"},  
  {"num_pairs",13,0,1,0,1933,0,0,0,0,0,0,"{key to apportionment among bin-based histogram variables} VarCommands.html#VarCAUV_Bin_Histogram"},  
  {"ordinates",14,0,3,2,1937,0,0,0,0,0,0,"{sets of ordinates for bin-based histogram variables} VarCommands.html#VarCAUV_Bin_Histogram"}  
}

12.1.4.305 static KeyWord kw_302 () [static]

Initial value:
12.1.4.306 static KeyWord kw_303 () [static]

Initial value:

{
  "descriptors",15,0,4,0,1979,0,0,0,0,0,"{Descriptors} VarCommands.html#VarDAUV_Hypergeometric",0,"hypergeometric_uncertain"},
  "num_drawn",13,0,3,3,1977,0,0,0,0,0,"{hypergeometric uncertain num drawn} VarCommands.html#VarDAUV_Hypergeometric",0,"hypergeometric_uncertain"},
  "selected_population",13,0,2,2,1975,0,0,0,0,0,"{hypergeometric uncertain selected_population} VarCommands.html#VarDAUV_Hypergeometric",0,"hypergeometric_uncertain"},
  "total_population",13,0,1,1,1973,0,0,0,0,0,"{hypergeometric uncertain total_population} VarCommands.html#VarDAUV_Hypergeometric",0,"hypergeometric_uncertain"}
}

12.1.4.307 static KeyWord kw_304 () [static]

Initial value:

{
  "lnuv_zetas",6,0,1,1,1840,0,0,0,0,0,0,0,0,"lognormal_uncertain"},
  "zetas",14,0,1,1,1841,0,0,0,0,0,0,0,0,"{lognormal uncertain zetas} VarCommands.html#VarCAUV_Lognormal",0,"lognormal_uncertain"
}

12.1.4.308 static KeyWord kw_305 () [static]

Initial value:

{
  "error_factors",14,0,1,1,1847,0,0,0,0,0,0,0,0,"{CHOOSE variance spec.}{lognormal uncertain error factors} VarCommands.html#VarCAUV_Lognormal",0,"lognormal_uncertain"}
12.1.4.309 static KeyWord kw_306 () [static]

Initial value:

{
    "descriptors",15,0,4,0,1853,0,0,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV_Lognormal",0,"lognormal_uncertain"},
    "lambdas",14,2,1,1,1839,kw_304,0,0,0,0,0,"{CHOOSE characterization (lognormal uncertain lambdas) VarCommands.html#VarCAUV_Lognormal",0,"lognormal_uncertain"},
    "lnuv_descriptors",7,0,4,0,1852,0,0,0,0,0,0,"lognormal_uncertain"},
    "lnuv_lambdas",6,0,2,0,1848,0,0,0,0,0,0,"lognormal_uncertain"},
    "lnuv_means",6,4,1,1,1842,kw_305,0,0,0,0,0,0,"lognormal_uncertain"},
    "lnuv_upper_bounds",6,0,3,0,1850,0,0,0,0,0,0,"lognormal_uncertain"},
    "lower_bounds",14,0,2,0,1849,0,0,0,0,0,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Lognormal",0,"lognormal_uncertain"},
    "means",14,4,1,1,1843,kw_305,0,0,0,0,"{lognormal uncertain means) VarCommands.html#VarCAUV_Lognormal",0,"lognormal_uncertain"},
    "upper_bounds",14,0,3,0,1851,0,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Lognormal",0,"lognormal_uncertain"}
}
12.1.4.311 static KeyWord kw_308 () [static]

Initial value:

```c
{
  "descriptors":15,0,3,0,1963,0,0,0,0,"{Descriptors} VarCommands.html#VarDAUV_NEGATIVE_BINOMIAL",0,"negative_binomial_uncertain"),
  "num_trials":13,0,2,2,1961,0,0,0,0,"{negative binomial uncertain success num_trials} VarCommands.html#VarDAUV_NEGATIVE_BINOMIAL",0,"negative_binomial_uncertain"),
  "prob_per_trial":6,0,1,1,1958,0,0,0,0,0,"{negative binomial uncertain prob_per_trial"},
  "probability_per_trial":14,0,1,1,1959,0,0,0,0,0,0,"{negative_binomial_uncertain"}
}
```

12.1.4.312 static KeyWord kw_309 () [static]

Initial value:

```c
{
  "descriptors":15,0,5,0,1835,0,0,0,0,"{Descriptors} VarCommands.html#VarCAUV_NORMAL",0,"normal_uncertain"),
  "lower_bounds":14,0,3,0,1831,0,0,0,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_NORMAL",0,"normal_uncertain"),
  "means":14,0,1,1,1827,0,0,0,0,"{normal uncertain means} VarCommands.html#VarCAUV_NORMAL",0,"normal_uncertain"),
  "nuv_descriptors":7,0,5,0,1834,0,0,0,0,0,0,"{normal uncertain nuv_descriptors"},
  "nuv_lower_bounds":6,0,3,0,1830,0,0,0,0,0,0,"{normal_uncertain nuv_lower_bounds"},
  "nuv_means":6,0,1,1,1826,0,0,0,0,0,0,"{normal_uncertain nuv_means"},
  "nuv_std_deviations":6,0,2,2,1828,0,0,0,0,0,0,"{normal_uncertain nuv_std_deviations"},
  "nuv_upper_bounds":6,0,4,0,1832,0,0,0,0,0,0,"{normal_uncertain nuv_upper_bounds"},
  "std_deviations":14,0,2,2,1829,0,0,0,0,0,0,"{normal uncertain std_deviations} VarCommands.html#VarCAUV_NORMAL",0,"normal_uncertain"),
  "upper_bounds":14,0,4,0,1833,0,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_NORMAL",0,"normal_uncertain")
```

12.1.4.313 static KeyWord kw_310 () [static]

Initial value:

```c
{
  "descriptors":15,0,2,0,1947,0,0,0,0,"{Descriptors} VarCommands.html#VarDAUV_POISSON",0,"poisson_uncertain"),
  "lambdas":14,0,1,1,1945,0,0,0,0,0,"{poisson uncertain lambdas} VarCommands.html#VarDAUV_POISSON",0,"poisson_uncertain")
```
12.1.4.314 static KeyWord kw_311 () [static]

Initial value:

```
{
    "descriptors",15,0,4,0,1879,0,0.,0.,0.,0,"{Descriptors} VarCommands.html#VarCAUV_Triangular",0,"triangular_uncertain"),
    "lower_bounds",14,0,2,2,1875,0,0.,0.,0.,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Triangular",0,"triangular_uncertain"),
    "modes",14,0,1,1,1873,0,0.,0.,0.,0,"{triangular uncertain modes} VarCommands.html#VarCAUV_Triangular",0,"triangular_uncertain"),
    "tuv_descriptors",7,0,4,0,1878,0,0.,0.,0,0,0,0,0,"triangular_uncertain"),
    "tuv_lower_bounds",6,0,2,2,1874,0,0.,0.,0,0,0,0,0,"triangular_uncertain"),
    "tuv_modes",6,0,1,1,1872,0,0.,0.,0,0,0,0,0,"triangular_uncertain"),
    "tuv_upper_bounds",6,0,3,3,1876,0,0.,0.,0,0,0,0,0,"triangular_uncertain"),
    "upper_bounds",14,0,3,3,1877,0,0.,0.,0,0,0,0,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Triangular",0,"triangular_uncertain"}
```

12.1.4.315 static KeyWord kw_312 () [static]

Initial value:

```
{
    "descriptors",15,0,3,0,1861,0,0.,0.,0.,0,"{Descriptors} VarCommands.html#VarCAUV_Uniform",0,"uniform_uncertain"),
    "lower_bounds",14,0,1,1,1857,0,0.,0.,0.,0,"{Distribution lower bounds} VarCommands.html#VarCAUV_Uniform",0,"uniform_uncertain"),
    "upper_bounds",14,0,2,2,1859,0,0.,0.,0.,0,"{Distribution upper bounds} VarCommands.html#VarCAUV_Uniform",0,"uniform_uncertain"),
    "uuv_descriptors",7,0,3,0,1860,0,0.,0.,0,0,0,0,0,"uniform_uncertain"),
    "uuv_lower_bounds",6,0,1,1,1856,0,0.,0.,0,0,0,0,0,"uniform_uncertain"),
    "uuv_upper_bounds",6,0,2,2,1858,0,0.,0.,0,0,0,0,0,"uniform_uncertain")
```

12.1.4.316 static KeyWord kw_313 () [static]

Initial value:

```
{
    "alphas",14,0,1,1,1925,0,0.,0.,0.,0,"{weibull uncertain alphas} VarCommands.html#VarCAUV_Weibull",0,"weibull_uncertain"),
    "betas",14,0,2,2,1927,0,0.,0.,0.,0,"{weibull uncertain betas} VarCommands.html#VarCAUV_Weibull",0,"weibull_uncertain"),
    "descriptors",15,0,3,0,1929,0,0.,0.,0.,0,"{Descriptors} VarCommands.html#VarCAUV_Weibull",0,"weibull_uncertain"),
    "wuv_alphas",6,0,1,1,1924,0,0.,0.,0,0,0,0,0,"weibull_uncertain"),
    "wuv_betas",6,0,2,2,1926,0,0.,0.,0,0,0,0,0,"weibull_uncertain"),
```
12.1.4.317 static KeyWord kw_315 () [static]

Initial value:

```
{"interface",0x308,8,5,5,2077,kw_14,0.,0.,0.,0,"{Interface} An interface specifies how function evaluations will be performed in order to map a set of parameters into a set of responses. InterfCommands.html"},
{"method",0x308,86,2,2,65,kw_220,0.,0.,0.,0,"{Method} A method specifies the name and controls of an iterative procedure, e.g., a sensitivity analysis, uncertainty quantification, or optimization method. MethodCommands.html"},
{"model",8,7,3,3,1543,kw_249,0.,0.,0.,0,"{Model} A model consists of a model type and maps specified variables through an interface to generate responses. ModelCommands.html"},
{"responses",0x308,19,6,6,2189,kw_267,0.,0.,0.,0,"{Responses} A responses object specifies the data that can be returned to DAKOTA through the interface after the completion of a function evaluation. RespCommands.html"},
{"strategy",0x108,10,1,1,kw_280,0.,0.,0.,0,"{Strategy} The strategy specifies the top level technique which will govern the management of iterators and models in the solution of the problem of interest. StratCommands.html"},
{"variables",0x308,37,4,4,1759,kw_314,0.,0.,0.,0,"{Variables} A variables object specifies the parameter set to be iterated by a particular method. VarCommands.html"}
```

12.1.4.318 KeyWord kw_316[10] [static]

Initial value:

```
{"abscissas",14,0,2,1,0,0.,0.,0,N_vam(newrvec,Var_Info_hba)},
{"counts",14,0,3,2,0,0.,0.,0,N_vam(newrvec,Var_Info_hbo)},
{"descriptors",15,0,4,0,0,0.,0.,0,N_vae(caulbl,CAUVar_histogram_bin)},
{"huv_bin_abscissas",6,0,2,1,0,0.,0.,0,-3,N_vam(newrvec,Var_Info_hba)},
{"huv_bin_counts",6,0,3,2,0,0.,0.,0,-3,N_vam(newrvec,Var_Info_hbo)},
{"huv_bin_descriptors",7,0,4,0,0,0.,0.,0,-3,N_vae(caulbl,CAUVar_histogram_bin)},
{"huv_bin_ordinates",14,0,3,2,0,0.,0.,0,N_vam(newrvec,Var_Info_hbo)},
{"huv_num_bin_pairs",5,0,1,0,0.,0.,0,1,N_vam(newiarray,Var_Info_nhbp)},
{"num_pairs",13,0,1,0,0.,0.,0,0,N_vam(newiarray,Var_Info_nhbp)},
{"ordinates",14,0,3,2,0,0.,0.0,N_vam(newrvec,Var_Info_hbo)}}
```

12.1.4.319 KeyWord kw_317[8] [static]

Initial value:
12.1 Dakota Namespace Reference

{  
  {"abscissas",14,0,2,1,0,0,0,0,0,N_vam(newrvec,Var_Info_hpa)},  
  {"counts",14,0,3,2,0,0,0,0,0,N_vam(newrvec,Var_Info_hpc)},  
  {"descriptors",15,0,4,0,0,0,0,0,0,N_vae(dauilbl,DAURVar_histogram_point)},  
  {"huv_num_point_pairs",5,0,1,0,0,0,0,4,N_vam(newiarray,Var_Info_nhpp)},  
  {"huv_point_abscissas",6,0,2,1,0,0,0,-4,N_vam(newrvec,Var_Info_hpa)},  
  {"huv_point_counts",6,0,3,2,0,0,0,-4,N_vam(newrvec,Var_Info_hpc)},  
  {"huv_point_descriptors",7,0,4,0,0,0,-4,N_vae(dauilbl,DAURVar_histogram_point)},  
  {"num_pairs",13,0,1,0,0,0,0,0,N_vam(newiarray,Var_Info_nhpp)}
}

12.1.4.320 KeyWord kw_318[4] [static]

Initial value:

{  
  {"descriptors",15,0,4,0,0,0,0,0,0,N_vae(dauilbl,DAUIVar_hypergeometric)},  
  {"num_drawn",13,0,3,3,0,0,0,0,0,N_vam(ivec,hyperGeomUncNumDrawn)},  
  {"selected_population",13,0,2,2,0,0,0,0,0,N_vam(ivec,hyperGeomUncSelectedPop)},  
  {"total_population",13,0,1,1,0,0,0,0,0,N_vam(ivec,hyperGeomUncTotalPop)}
}

12.1.4.321 KeyWord kw_319[2] [static]

Initial value:

{  
  {"lnuv_zetas",6,0,1,1,0,0,0,1,1,N_vam(RealLb,lognormalUncZetas)},  
  {"zetas",14,0,1,1,0,0,0,0,0,N_vam(RealLb,lognormalUncZetas)}
}

12.1.4.322 KeyWord kw_320[4] [static]

Initial value:

{  
  {"error_factors",14,0,1,1,0,0,0,0,0,N_vam(RealLb,lognormalUncErrFacts)},  
  {"lnuv_error_factors",6,0,1,1,0,0,0,0,-1,N_vam(RealLb,lognormalUncErrFacts)},  
  {"lnuv_std_deviations",6,0,1,1,0,0,0,0,1,N_vam(RealLb,lognormalUncStdDevs)},  
  {"std_deviations",14,0,1,1,0,0,0,0,0,N_vam(RealLb,lognormalUncStdDevs)}
}
12.1.4.323  KeyWord kw_321[10]  [static]

Initial value:

```
{ "descriptors",15,0,4,0,0,0.,0.,0,N_vae(caulbl,CAUVar_lognormal) },
{ "lambdas",14,2,1,1,kw_319,0.,0.,0.,0,N_vam(rvec,lognormalUncLambdas),
  "lnuv_descriptors",7,0,4,0,0.,0.,-2,N_vae(caulbl,CAUVar_lognormal),
  "lnuv_lambdas",6,2,1,1,kw_319,0.,0.,-2,N_vam(rvec,lognormalUncLambdas),
  "lnuv_lower_bounds",6,0,2,0,0,0.,0.,3,N_vam(RealLb,lognormalUncLowerBnds),
  "lnuv_means",6,4,1,1,kw_320,0.,0.,3,N_vam(RealLb,lognormalUncMeans),
  "lnuv_upper_bounds",6,0,3,0,0,0.,0.,3,N_vam(RealUb,lognormalUncUpperBnds),
  "lower_bounds",14,0,2,0,0,0.,0.,0,N_vam(RealLb,lognormalUncLowerBnds),
  "means",14,4,1,1,kw_320,0.,0.,3,N_vam(RealLb,lognormalUncMeans),
  "upper_bounds",14,0,3,0,0,0.,0.,0,N_vam(RealUb,lognormalUncUpperBnds)}
```

12.1.4.324  KeyWord kw_322[6]  [static]

Initial value:

```
{ "descriptors",15,0,3,0,0,0.,0.,0,N_vae(caulbl,CAUVar_loguniform) },
{ "lower_bounds",14,0,1,1,0,0,0.,0.,0,N_vam(RealLb,loguniformUncLowerBnds),
  "luuv_descriptors",7,0,3,0,0,0.,0.,-2,N_vae(caulbl,CAUVar_loguniform),
  "luuv_lower_bounds",6,0,1,1,0,0,0.,0.,-2,N_vam(RealLb,loguniformUncLowerBnds),
  "luuv_upper_bounds",6,0,2,2,0,0,0.,0.,1,N_vam(RealUb,loguniformUncUpperBnds),
  "upper_bounds",14,0,2,2,0,0,0.,0.,0,N_vam(RealUb,loguniformUncUpperBnds)}
```

12.1.4.325  KeyWord kw_323[4]  [static]

Initial value:

```
{ "descriptors",15,0,3,0,0,0.,0.,0,N_vae(dauilbl,DAUIVar_negative_binomial) },
{ "num_trials",13,0,2,2,0,0,0.,0.,0,N_vam(ivec,negBinomialUncNumTrials),
  "prob_per_trial",6,0,1,1,0,0,0.,0.,1,N_vam(rvec,negBinomialUncProb}
12.1 Dakota Namespace Reference

12.1.4.326 KeyWord kw_324[10] [static]

Initial value:

```dakota
{
    "descriptors",15,0,5,0,0,0,0,0,0,N_vae( caulbl, CAUVar_normal )
    "lower_bounds",14,0,3,0,0,0,0,0,0,N_vam( rvec, normalUncLowerBnds )
    "means",14,0,1,1,0,0,0,0,0,N_vam( rvec, normalUncMeans )
    "nuv_descriptors",7,0,5,0,0,0,0,-3,N_vae( caulbl, CAUVar_normal )
    "nuv_lower_bounds",6,0,3,0,0,0,0,-3,N_vam( rvec, normalUncLowerBnds )
    "nuv_means",6,0,1,1,0,0,0,0,-3,N_vam( rvec, normalUncMeans )
    "nuv_std_deviations",6,0,2,2,0,0,0,0,2,N_vam( RealLb, normalUncStdDevs )
    "nuv_upper_bounds",6,0,4,0,0,0,0,-3,N_vam( rvec, normalUncUpperBnds )
    "std_deviations",14,0,1,0,0,0,0,0,0,N_vam( rvec, normalUncStdDevs )
    "upper_bounds",14,0,4,0,0,0,0,0,0,N_vam( rvec, normalUncUpperBnds )
}
```

12.1.4.327 KeyWord kw_325[2] [static]

Initial value:

```dakota
{
    "descriptors",15,0,2,0,0,0,0,0,0,N_vae( dauilbl, DAUIVar_poisson )
    "lambdas",14,0,1,1,0,0,0,0,0,N_vam( rvec, poissonUncLambdas )
}
```

12.1.4.328 KeyWord kw_326[8] [static]

Initial value:

```dakota
{
    "descriptors",15,0,4,0,0,0,0,0,0,N_vae( caulbl, CAUVar_triangular )
    "lower_bounds",14,0,2,2,0,0,0,0,0,N_vam( RealLb, triangularUncLowerBnds )
    "modes",14,0,1,1,0,0,0,0,0,N_vam( rvec, triangularUncModes )
    "tuv_descriptors",7,0,4,0,0,0,0,-3,N_vae( caulbl, CAUVar_triangular )
    "tuv_lower_bounds",6,0,2,2,0,0,0,-3,N_vam( RealLb, triangularUncLowerBnds )
    "tuv_modes",6,0,1,1,0,0,0,-3,N_vam( rvec, triangularUncModes )
}
```
Namespace Documentation

12.1.4.329 KeyWord kw_327[6] [static]
Initial value:

{"descriptors",15,0,3,0,0,0,0,0,0,N_vae(caulbl,CAUVar_uniform)},
{"lower_bounds",14,0,1,1,0,0,0,0,0,N_vam(RealLb,uniformUncLowerBnds)},
{"upper_bounds",14,0,2,2,0,0,0,0,0,N_vam(RealUb,uniformUncUpperBnds)},
{"uuv_descriptors",7,0,3,0,0,0,0,0,0,3,N_vae(caulbl,CAUVar_uniform)},
{"uuv_lower_bounds",6,0,1,1,0,0,0,0,0,3,N_vam(RealLb,uniformUncLowerBnds)},
{"uuv_upper_bounds",6,0,2,2,0,0,0,0,0,3,N_vam(RealUb,uniformUncUpperBnds)}

12.1.4.330 KeyWord kw_328[6] [static]
Initial value:

{"alphas",14,0,1,1,0,0,0,0,0,N_vam(RealLb,weibullUncAlphas)},
{"betas",14,0,2,2,0,0,0,0,0,N_vam(RealLb,weibullUncBetas)},
{"descriptors",15,0,3,0,0,0,0,0,0,N_vae(caulbl,CAUVar_weibull)},
{"wuv_alphas",6,0,1,1,0,0,0,0,0,3,N_vam(RealLb,weibullUncAlphas)},
{"wuv_betas",6,0,2,2,0,0,0,0,0,3,N_vam(RealLb,weibullUncBetas)},
{"wuv_descriptors",7,0,3,0,0,0,0,0,0,3,N_vae(caulbl,CAUVar_weibull)}

12.1.4.331 KeyWord kw_330[6] [static]
Initial value:

{"interface",0x308,8,5,5,kw_14,0,0,0,0,N_ifm3(start,0,stop)},
{"method",0x308,86,2,2,kw_234,0,0,0,N_mdm3(start,0,stop)},
{"model",8,7,3,3,kw_264,0,0,0,0,N_mom3(start,0,stop)},
{"responses",0x308,19,6,6,kw_282,0,0,0,0,N_rem3(start,0,stop)},
{"strategy",0x108,10,1,1,kw_295,0,0,0,0,NIDRProblemDescDB::strategy_start},
{"variables",0x308,37,4,4,kw_329,0,0,0,0,N_vam3(start,0,stop)}
12.1.4.332  Var_uinfo CAUVLbl[CAUVar_Nkinds]  [static]

Initial value:

```
{ 
  VarLabelInfo(nuv_, NormalUnc), 
  VarLabelInfo(lnuv_, LognormalUnc), 
  VarLabelInfo(uuv_, UniformUnc), 
  VarLabelInfo(luuv_, LoguniformUnc), 
  VarLabelInfo(tuv_, TriangularUnc), 
  VarLabelInfo(euv_, ExponentialUnc), 
  VarLabelInfo(beuv_, BetaUnc), 
  VarLabelInfo(gauv_, GammaUnc), 
  VarLabelInfo(guuv_, GumbelUnc), 
  VarLabelInfo(fuv_, FrechetUnc), 
  VarLabelInfo(wuv_, WeibullUnc), 
  VarLabelInfo(hbuv_, HistogramBinUnc) 
}
```

12.1.4.333  Var_uinfo DAUVLbl[DAUVar_Nkinds]  [static]

Initial value:

```
{ 
  VarLabelInfo(puv_, PoissonUnc), 
  VarLabelInfo(biuv_, BinomialUnc), 
  VarLabelInfo(nbuv_, NegBinomialUnc), 
  VarLabelInfo(geuv_, GeometricUnc), 
  VarLabelInfo(hguv_, HyperGeomUnc) 
}
```

12.1.4.334  Var_uinfo DAURVLbl[DAURVar_Nkinds]  [static]

Initial value:

```
{ 
  VarLabelInfo(hpuv_, HistogramPtUnc) 
}
```

12.1.4.335  Var_uinfo CEUVLbl[CEUVar_Nkinds]  [static]

Initial value:

```
{ 
  VarLabelInfo(ciuv_, ContinuousIntervalUnc) 
}
```

12.1.4.336  Var_uinfo DEUVLbl[DEUVar_Nkinds]  [static]

Initial value:
{ VarLabelInfo(diuv_, DiscreteIntervalUnc),
  VarLabelInfo(dusiv_, DiscreteUncSetInt) }

12.1.4.337 Var_uinfo DEURVLbl[DEURVar_Nkinds] [static]

Initial value:

{ VarLabelInfo(dusrv_, DiscreteUncSetReal) }

12.1.4.338 Var_uinfo DiscSetLbl[DiscSetVar_Nkinds] [static]

Initial value:

{ VarLabelInfo(ddsiv_, DiscreteDesSetInt),
  VarLabelInfo(ddsrv_, DiscreteDesSetReal),
  VarLabelInfo(dssiv_, DiscreteStateSetInt),
  VarLabelInfo(dssrv_, DiscreteStateSetReal) }

12.1.4.339 VarLabelChk Vlch[] [static]

Initial value:

{ AVI numContinuousDesVars, AVI continuousDesignLabels, "cdv_", "cdv_descriptors" },
{ AVI numDiscreteDesRangeVars, AVI discreteDesignRangeLabels, "ddriv_", "ddriv_descriptors" },
{ AVI numDiscreteDesSetIntVars, AVI discreteDesignSetIntLabels, "ddsiv_", "ddsv_descriptors" },
{ AVI numDiscreteDesSetRealVars, AVI discreteDesignSetRealLabels, "ddsrv_", "ddsrv_descriptors" },
{ AVI numContinuousStateVars, AVI continuousStateLabels, "csv_", "csv_descriptors" },
{ AVI numDiscreteStateRangeVars, AVI discreteStateRangeLabels, "dsriv_", "dsriv_descriptors" },
{ AVI numDiscreteStateSetIntVars, AVI discreteStateSetIntLabels, "dssiv_", "dssiv_descriptors" },
{ AVI numDiscreteStateSetRealVars, AVI discreteStateSetRealLabels, "dssrv_", "dssrv_descriptors" },
{ AVI numContinuousDesVars, AVI continuousDesignScaleTypes, 0, "cdv_scale_types" }

12.1.4.340 VLreal VLR[N_VLR] [static]

Initial value:  

Dakota Version 5.4 Developers Manual generated on November 7, 2013
12.1 Dakota Namespace Reference

```
{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}
```

12.1.4.341 VLint VLI[N_VLI] [static]

Initial value:

```
{DAUIVar_Nkinds, AVI DAUIv, DAUVLbl, 
 DVR discreteIntAleatoryUncLabels, 
 DVR discreteIntAleatoryUncLowerBnds, 
 DVR discreteIntAleatoryUncUpperBnds, 
 DVR discreteIntAleatoryUncVars},
{DEUIVar_Nkinds, AVI DEUIv, DEUVLbl, 
 DVR discreteIntEpistemicUncLabels, 
 DVR discreteIntEpistemicUncLowerBnds, 
 DVR discreteIntEpistemicUncUpperBnds, 
 DVR discreteIntEpistemicUncVars}
```

12.1.4.342 Var_check var_mp_check_cv[] [static]

Initial value:

```
{Vchk_3(continuous_design,ContinuousDes), 
 Vchk_3(continuous_state,ContinuousState) }
```

12.1.4.343 Var_check var_mp_check_dset[] [static]

Initial value:

```
{Vchk_3(discrete_design_set_integer,DiscreteDesSetInt), 
 Vchk_3(discrete_design_set_real,DiscreteDesSetReal), 
 Vchk_3(discrete_state_set_integer,DiscreteStateSetInt), 
 Vchk_3(discrete_state_set_real,DiscreteStateSetReal) }
```
12.1.4.344  Var_check var_mp_check_cau[]  [static]

Initial value:

{       
    Vchk_3(normal_uncertain,NormalUnc),
    Vchk_3(lognormal_uncertain,LognormalUnc),
    Vchk_3(uniform_uncertain,UniformUnc),
    Vchk_3(loguniform_uncertain,LoguniformUnc),
    Vchk_3(triangular_uncertain,TriangularUnc),
    Vchk_3(exponential_uncertain,ExponentialUnc),
    Vchk_3(beta_uncertain,BetaUnc),
    Vchk_3(gamma_uncertain,GammaUnc),
    Vchk_3(gumbel_uncertain,GumbelUnc),
    Vchk_3(frechet_uncertain,FrechetUnc),
    Vchk_3(weibull_uncertain,WeibullUnc),
    Vchk_3(histogram_bin_uncertain,HistogramBinUnc) }

12.1.4.345  Var_check var_mp_check_daiui[]  [static]

Initial value:

{       
    Vchk_3(poisson_uncertain,PoissonUnc),
    Vchk_3(binomial_uncertain,BinomialUnc),
    Vchk_3(negative_binomial_uncertain,NegBinomialUnc),
    Vchk_3(geometric_uncertain,GeometricUnc),
    Vchk_3(hypergeometric_uncertain,HyperGeomUnc) }

12.1.4.346  Var_check var_mp_check_daur[]  [static]

Initial value:

{       
    Vchk_3(histogram_point_uncertain,HistogramPtUnc) }

12.1.4.347  Var_check var_mp_check_ceu[]  [static]

Initial value:

{       
    Vchk_3(continuous_interval_uncertain,ContinuousIntervalUnc) }

12.1.4.348  Var_check var_mp_check_deui[]  [static]

Initial value:

{       
    Vchk_3(discrete_interval_uncertain,DiscreteIntervalUnc),
    Vchk_3(discrete_uncertain_set_integer,DiscreteUncSetInt) }
12.1.4.349  Var_check var_mp_check_deur[] [static]
Initial value:

{  
    Vchk_3(discrete_uncertain_set_real,DiscreteUncSetReal)  
}

12.1.4.350  Var_rcheck var_mp_cbound[] [static]
Initial value:

{  
    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(loguniform_uncertain,LoguniformUnc,loguniformUnc),  
    Vchk_5(triangular_uncertain,TriangularUnc,triangularUnc),  
    Vchk_5(beta_uncertain,BetaUnc,betaUnc)  
}

12.1.4.351  Var_icheck var_mp_drange[] [static]
Initial value:

{  
    Vchk_7(discrete_design_range,DiscreteDesRange,discreteDesignRange),  
    Vchk_7(discrete_state_range,DiscreteStateRange,discreteStateRange)  
}

12.1.4.352  const char∗ SCI_FIELD_NAMES[]
Initial value:

{  
    "dakota_type", "numFns", "numVars", "numACV", "numADIV",  
    "numADRV", "numDerivVars", "xC", "xDI",  
    "xDR", "xCLabels", "xDILabels",  
    "xDRLabels", "directFnASV", "directFnASM",  
    "directFnDVV", "directFnDVV_bool",  
    "fnFlag", "gradFlag", "hessFlag",  
    "fnVals", "fnGrads", "fnHessians",  
    "fnLabels", "failure", "currEvalId"  
}

fields to pass to Scilab in Dakota structure
Referenced by ScilabInterface::scilab_engine_run().

12.1.4.353  const int SCI_NUMBER_OF_FIELDS = 26

number of fields in above 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 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 (BiStream &s)
  read an active set object from the binary restart stream

• void write (BoStream &s) const
  write an active set object to the binary restart stream

• void read (MPIUnpackBuffer &s)
  read an active set object from a packed MPI buffer
13.1 ActiveSet Class Reference

- void write (MPIPackBuffer &s) const
  write an active set object to a packed MPI buffer

Private Attributes

- ShortArray requestVector
  the vector of response requests

- SizetArray derivVarsVector
  the vector of variable ids used for computing derivatives

Friends

- 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 DVV 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

13.1.2.1 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=(), Dakota::operator==(), ActiveSet::read(), ActiveSet::request_value(), ActiveSet::request_values(), ActiveSet::request_vector(), ActiveSet::reshape(), ActiveSet::write(), and ActiveSet::write_annotated().

13.1.2.2 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=(), Dakota::operator==(), ActiveSet::read(), ActiveSet::reshape(), ActiveSet::write(), and ActiveSet::write_annotated().

The documentation for this class was generated from the following files:

- DakotaActiveSet.hpp
- DakotaActiveSet.cpp
13.2 Analyzer Class Reference

Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy. Inheritance diagram for Analyzer::

```
Analyzer
  Iterator
  NonD
  PStudy
  DACE
  Verification
  EfficientSubspaceMethod
  NonDCalibration
  NonDExpansion
  NonDIntegration
  NonDInterval
  NonDPofDarts
  NonDReliability
  NonDSampling
  PSUADEDesignCompExp
  FSDK DesignCompExp
  ParamStudy
  PStUDesignCompExp
  RichExtrapVerification
```

**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`

- `virtual void vary_pattern (bool pattern_flag)`
  `sets varyPattern in derived classes that support it`

**Protected Member Functions**

- `Analyzer ()`
  `default constructor`

- `Analyzer (Model &model)`
  `standard constructor`
• **Analyzer (NoDBBaseConstructor, Model &model)**
  alternate constructor for instantiations "on the fly" with a Model

• **Analyzer (NoDBBaseConstructor)**
  alternate constructor for instantiations "on the fly" without a Model

• ~Analyzer ()
  destructor

• virtual void get_parameter_sets (Model &model)
  Returns one block of samples (ndim * num_samples).

• 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

• void pre_output()

• void print_results (std::ostream &s)
  print the final iterator results

• 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 ()
  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.
13.2 Analyzer Class Reference

- **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 variance_based_decomp (int ncont, int ndiscint, int ndiscreal, int num_samples)**

- **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 sample_to_variables (const Real *sample_c_vars, Variables &vars)**
  
  convert samples array to variables array; e.g., allSamples to allVariables

- **void samples_to_variables_array (const RealMatrix &sample_matrix, VariablesArray &vars_array)**
  
  convert samples array to variables array; e.g., allSamples to allVariables

- **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

- **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

- Real **vbdDropTol**
  
  *tolerance for omitting output of small VBD indices*

- RealVectorArray **S4**
  
  *VBD main effect indices.*

- RealVectorArray **T4**
  
  *VBD total effect indices.*

13.2.1 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.

13.2.2 Member Function Documentation

13.2.2.1 **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**, **Model::current_response()**, **Model::current_variables()**, **Iterator::iteratedModel**, **Iterator::outputLevel**, **Model::parallel_library()**, **Dakota::write_data_tabular()**, **Dakota::write_precision**, and **Iterator::writePrecision**.
13.2.2.2  void print_results (std::ostream & s)  [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 PStudyDACE, Verification, NonDAadaptImpSampling, NonDAadaptiveSampling, NonDExpansion, NonDGlocalReliability, NonDGPImpSampling, NonDIncrelmHS Sampling, NonDInterval, NonDLHSSampling, NonDLocalReliability, NonDPDOFDarts, and RichExtrapVerification.

References Analyzer::bestVarsRespMap, ParamResponsePair::eval_id(), Response::function_values(), Analyzer::numLsqTerms, Analyzer::numObjFns, ParamResponsePair::prp_parameters(), ParamResponsePair::prp_response(), and Dakota::write_data_partial().

13.2.2.3  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::activeSet, Analyzer::allHeaders, Analyzer::allResponses, Analyzer::allSamples, Analyzer::allVariables, Model::asynch_compute_response(), Iterator::asynchFlag, Analyzer::compactMode, Model::compute_response(), Response::copy(), Model::current_response(), Model::current_variables(), Model::evaluation_id(), Model::synchronize(), Analyzer::update_best(), Analyzer::update_model_from_sample(), and Analyzer::update_model_from_variables().

Referenced by NonDSparseGrid::evaluate_grid_increment(), NonDSparseGrid::evaluate_set(), PSUADesignCompExp::extract_trends(), ParamStudy::extract_trends(), FSUADesignCompExp::extract_trends(), DDACEDesignCompExp::extract_trends(), NonDLHSSampling::quantify_uncertainty(), NonDIntegration::quantify_uncertainty(), NonDIncrelmHSSampling::quantify_uncertainty(), NonDAadaptImpSampling::quantify_uncertainty(), and Analyzer::variance_based_decomp().

13.2.2.4  void variance_based_decomp (int ncont, int ndiscint, int ndiscreal, int num_samples)  [protected]

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::allResponses, Analyzer::allSamples, Analyzer::allVariables, Analyzer::compactMode, Variables::continuous_variables(), Dakota::copy_data(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Analyzer::evaluate_parameter_sets(), Analyzer::get_parameter_sets(), Iterator::iteratedModel, Iterator::numFunctions, Analyzer::S4, Analyzer::T4, and Analyzer::vary_pattern().

Referenced by FSUADesignCompExp::extract_trends(), DDACEDesignCompExp::extract_trends(), and NonDLHSSampling::quantify_uncertainty().
13.2.2.5 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(), Iterator::iteratedModel, Iterator::outputLevel, Model::parallel_library(), and Analyzer::update_best().

Referenced by PSUADEDesignCompExp::post_input(), ParamStudy::post_input(), NonDLHSSampling::post_input(), FSUDesignCompExp::post_input(), and DDACEDesignCompExp::post_input().

13.2.2.6 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, Iterator::numContinuousVars, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, Iterator::numFunctions, Model::response_labels(), Analyzer::S4, Analyzer::T4, Analyzer::vbdDropTol, and Dakota::write_precision.

Referenced by NonDLHSSampling::print_results(), and PStudyDACE::print_results().

The documentation for this class was generated from the following files:

- DakotaAnalyzer.hpp
- DakotaAnalyzer.cpp
13.3 ApplicationInterface Class Reference

Derived class within the interface class hierarchy for supporting interfaces to simulation codes. Inheritance diagram for ApplicationInterface::

```
Interface
  ApplicationInterface
    DirectApplicInterface
    ProcessApplicInterface
      MatlabInterface
      PythonInterface
      ScilabInterface
      TestDriverInterface
      ParallelDirectApplicInterface
      SerialDirectApplicInterface
      ProcessHandleApplicInterface
      SysCallApplicInterface
```

Public Member Functions

- **ApplicationInterface** (const ProblemDescDB &problem db)
  
  *constructor*

- **~ApplicationInterface** ()
  
  *destructor*

Protected Member Functions

- void **init_communicator**s (const IntArray &message_lengths, int max_iterator_concurrency)
  
  *allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.*

- void **set_communicator**s (const IntArray &message_lengths, int max_iterator_concurrency)
  
  *set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).*

- void **free_communicator**s ()
  
  *deallocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.*

- void **init_serial** ()

- int **asynch_local_evaluation_concurrency** () const
return asynchLocalEvalConcurrency

- String interface_synchronization () const
  return interfaceSynchronization

- bool evaluation_cache () 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 & synch ()
  executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns all jobs

- const IntResponseMap & synch_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_iterator_concurrency)
  checks on asynchronous configuration (for direct interfaces)

- bool check_multiprocessor_asynchronous (bool warn, int max_iterator_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.
13.3 ApplicationInterface Class Reference

- **virtual 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.

- **virtual 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.

- **virtual void init_communicators_checks** (int max_iterator_concurrency)
  
  perform construct-time error checks on the parallel configuration

- **virtual void set_communicators_checks** (int max_iterator_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 concurrent 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

• int numAnalysisServersSpec
  user spec for number of analysis servers

• bool multiProcAnalysisFlag
  flag for multiprocessor analysis partitions

• bool asynchLocalAnalysisFlag
  flag for asynchronous local parallelism of analyses

• int asynchLocalAnalysisConcSpec
  user specification for asynchronous local analysis concurrency

• int asynchLocalAnalysisConcurrency
  limits the number of concurrent analyses in asynchronous local scheduling and specifies hybrid concurrency when message passing

• 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()

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 master_dynamic_schedule_evaluations ()
  blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a dedicated master partition; executes on iteratorComm master
13.3 ApplicationInterface Class Reference

- 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_dynamic_schedule_evaluations_nowait ()
  
  execute a nonblocking static/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 async local jobs from local_prp_queue, as limited by server capacity

- void assign_asynch_local_queue_nowait (PRPQueue &local_prp_queue, PRPQueuelter &local_prp_iter)
  
  helper function for updating an active local queue by backfilling async local jobs from local_prp_queue, as limited by server capacity

- size_t test_local_backfill (PRPQueue &assign_queue, PRPQueuelter &assign_iter)
  
  helper function for testing active async local jobs and then backfilling

- size_t test_receives_backfill (PRPQueuelter &assign_iter, bool peer_flag)
  
  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 at ie scheduling level

- int numEvalServers
  current number of evaluation servers

- int numEvalServersSpec
  user specification for number of evaluation servers

- bool eaMessagePass
  flag for message passing at ea scheduling level

- 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()

• String evalScheduling
  user specification of evaluation scheduling algorithm (self, static, or no spec). Used for manual overrides of the auto-configure logic in ParallelLibrary::resolve_inputs().

• String analysisScheduling
  user specification of analysis scheduling algorithm (self, static, or no spec). Used for manual overrides of the auto-configure logic in ParallelLibrary::resolve_inputs().

• int asynchLocalEvalConcSpec
  user specification for asynchronous local evaluation concurrency

• 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 asynch local static schedules)

• String interfaceSynchronization
  interface synchronization specification: synchronous (default) or asynchronous

• 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 restartFileFlag
  used to manage a user request to deactivate the restart file (i.e., insertions into write_restart).

• 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
13.3 ApplicationInterface Class Reference

- **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 evalIdCnt, map value is corresponding response.*

- **std::map< int, std::pair< PRPQueueHIter, Response >> beforeSynchDuplicateMap**  
  *used to bookkeep evalIdCnt, 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 synch() or synch_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 synch() or synch_nowait().*

- **PRPQueue asynchLocalActivePRPQueue**  
  *used by nonblocking asynchronous local schedulers to bookkeep active local jobs*

- **std::map< int, IntIntPair > msgPassRunningMap**  
  *used by nonblocking message passing schedulers to bookkeep which jobs are running remotely*

- **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*

### 13.3.1 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.
13.3.2 Member Function Documentation

13.3.2.1 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().

13.3.2.2 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_compute_response() and derived_asynch_compute_response() 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 synch() or synch_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(), Interface::interfaceId, ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, Interface::newEvalIdCntr, Interface::newFnGradCounter, Interface::newFnHessCounter, Interface::newFnValCounter, Interface::outputLevel, ActiveSet::request_vector(), Interface::response_mapping(), ApplicationInterface::restartFileFlag, and Dakota::write_restart.

13.3.2.3 const IntResponseMap & synch () [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, ApplicationIn-
13.3 ApplicationInterface Class Reference

interface::beforeSynchDuplicateMap, 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, and Interface::response_mapping().

13.3.2.4 const IntResponseMap & synch_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 Dakota::abort_handler(), Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::asynchronous_local_evaluations_nowait(), ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, Interface::coreMappings, ParamResponsePair::eval_id(), ApplicationInterface::evalScheduling, 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(), ParamResponsePair::prp_response(), Interface::rawResponseMap, Interface::response_mapping(), and Response::update().

13.3.2.5 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().

13.3.2.6 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(), Interface::ieDedMasterFlag,
13.3.2.7 **void init_communicators_checks (int max_iterator_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, ProcessHandleApplicInterface, and SysCallApplicInterface.

Referenced by ApplicationInterface::init_communicators().

13.3.2.8 **void set_communicators_checks (int max_iterator_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, ParallelDirectApplicInterface, SerialDirectApplicInterface, ProcessHandleApplicInterface, and SysCallApplicInterface.

Referenced by ApplicationInterface::set_communicators().

13.3.2.9 **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 SysCallApplicInterface::create_evaluation_process(), ProcessHandleApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().

13.3.2.10 **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(), ParallelLibrary::waitall().
13.3 ApplicationInterface Class Reference

ApplicationInterface::synchronous_local_analysis(), and ParallelLibrary::wait().

Referenced by SysCallApplicInterface::create_evaluation_process(), ProcessHandleApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().

13.3.2.11 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, Interface::evalIdCntr, Dakota::hashedQueueEnd(), ApplicationInterface::historyDuplicateMap, Interface::interfaceId, Dakota::lookup_by_val(), and Response::update().

Referenced by ApplicationInterface::map().

13.3.2.12 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 synch() 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

References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, Dakota::lookup_by_eval_id(), ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::parallelLib, ApplicationInterface::receive_evaluation(), ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by ApplicationInterface::synch().

13.3.2.13 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 `synch()` 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::synch()`.

### 13.3.2.14 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 `synch()` 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::synch()`.

### 13.3.2.15 void `asynchronous_local_evaluations()` [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 `synch()` 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::asynchLocalEvalConcurrency`, `ApplicationInterface::beforeSynchCorePRPQueue`, `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::synch()`.
13.3 ApplicationInterface Class Reference

face::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::synch().

13.3.2.16 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().

13.3.2.17 void master_dynamic_schedule_evaluations_nowait () [private]

execute a nonblocking dynamic schedule in a master-slave partition This code is called from synch_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::msgPassRunningMap, ApplicationInterface::numEvalServers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synch_nowait().

13.3.2.18 void peer_dynamic_schedule_evaluations_nowait () [private]

execute a nonblocking static/dynamic schedule in a peer partition This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synch_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. 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 Dakota::abort_handler(), ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, 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::synch_nowait().

13.3.2.19 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 asynch case (background system call, nonblocking fork, or threads). It is called from synch_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 synch_nowait().

Also supports asynchronous local evaluations with static scheduling. This scheduling policy specifically ensures that a completed asynchronous evaluation eval_id is replaced with an equivalent one, modulo asynchLocalEvalConcurrency. 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::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchLocalEvalStatic, and ApplicationInterface::test_local_backfill().

Referenced by ApplicationInterface::synch_nowait().

13.3.2.20 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 Dakota::array_write_annotated(), ParallelLibrary::bcast_e(), Interface::currEvalId, ApplicationInterface::derived_map(), ApplicationInterface::evalCommRank, ParallelLibrary::isend_ie(), ApplicationInterface::lenResponseMessage, ApplicationInterface::lenVarsActSetMessage, ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, ApplicationInterface::parallelLib, ParallelLibrary::recv_ie(), MPIPackBuffer::reset(), and ParallelLibrary::wait().

Referenced by ApplicationInterface::serve_evaluations().
13.3.2.21  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 Dakota::array_write.annotated(), ParallelLibrary::bcast_e(), Interface::currEvalId,
ApplicationInterface::derived_map(), ApplicationInterface::lenVarsActSetMessage,
ApplicationInterface::manage_failure(), and ApplicationInterface::parallelLib.

Referenced by ApplicationInterface::serve_evaluations().

13.3.2.22  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(), ParallelLibrary::test(), and ApplicationInterface::test_local_evaluations().

Referenced by ApplicationInterface::serve_evaluations().

13.3.2.23  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 multiproces-
sor slave/peer servers. It matches the multiProcEvalFlag bcasts in ApplicationInterface::asynchronous_local_evaluations().

References Dakota::abort_handler(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ParallelLibrary::bcast_e(), ApplicationInterface::completionSet, ApplicationInterface::derived_map_asynch(), Interface::interfaceId, ApplicationInterface::lenVarsActSetMessage, Dakota::lookup_by_eval_id(), ApplicationInterface::parallelLib, MPIUnpackBuffer::reset(), 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
13.4 Approximation Class Reference

Base class for the approximation class hierarchy. Inheritance diagram for Approximation:

```
Approximation
  GaussProcApproximation
  PecosApproximation
  SurfpackApproximation
  TANA3Approximation
  TaylorApproximation
```

### Public Member Functions

- **Approximation ()**
  - default constructor

- **Approximation (ProblemDescDB &problem_db, size_t num_vars)**
  - standard constructor for envelope

- **Approximation (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  - alternate constructor

- **Approximation (const Approximation &approx)**
  - copy constructor

- virtual ~Approximation ()
  - destructor

- **Approximation operator= (const Approximation &approx)**
  - assignment operator

- virtual void build ()
  - builds the approximation from scratch

- virtual void rebuild ()
  - 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 restore ()
  - restores state prior to previous append()

- virtual bool restore_available ()
  - queries availability of restoration for trial set

- virtual size_t restoration_index ()
virtual void finalize ()
    finalize approximation by applying all remaining trial sets

virtual size_t finalization_index (size_t i)
    return index of i-th trailing trial set within restorable bookkeeping sets

virtual void store ()
    store current approximation for later combination

virtual void combine (short corr_type)
    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 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 void primary_diagnostics (int fn_index)
    compute and print all requested diagnostics and cross-validation

virtual void challenge_diagnostics (const RealMatrix &challenge_points, int fn_index)
    compute and print all requested diagnostics for user provided challenge pts

virtual const RealVector & approximation_coefficients () const
    return the coefficient array computed by build()/rebuild()

virtual void approximation_coefficients (const RealVector &approx_coeffs)
    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=false)
  
  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 samples to build the approximation type in `numVars` dimensions (default same as `min_points`)

• int `num_variables` () const
  
  return the number of variables used in the approximation

• 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)
  append to SurrogateData::respData or assign to SurrogateData::anchorResp

- 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 pop_count (size_t count)
  appends to popCountStack (number of entries to pop from end of SurrogateData::{vars,resp}Data, based on size of last data set appended)

- void clear_all ()
  clear all build data (current and history) to restore original state

- void clear_anchor ()
  clear SurrogateData::anchor{Vars,Resp}

- void clear_data ()
  clear SurrogateData::{vars,resp}Data

- void clear_saved ()
  clear popCountStack and SurrogateData::saved{Vars,Resp}Trials

- 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)

- 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, 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)

- Approximation (NoDBBaseConstructor, 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

- short outputLevel
  output verbosity level: [SILENT,QUIET,NORMAL,VERBOSE,DEBUG]_OUTPUT
• **int 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.*

• **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)*

• **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

- **Approximation * get_approx** (ProblemDescDB &problem_db, size_t num_vars)  
  *Used only by the standard envelope constructor to initialize approxRep to the appropriate derived type.*

- **Approximation * get_approx** (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 approxRep to the appropriate derived type.*
Private Attributes

- SizetArray popCountStack
  a stack managing the number of points previously added by calls to append() that can be removed by calls to pop()

- Approximation * approxRep
  pointer to the letter (initialized only for the envelope)

- int referenceCount
  number of objects sharing approxRep

13.4.1 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.

13.4.2 Constructor & Destructor Documentation

13.4.2.1 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 Approxima-
tion object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and
destructor.

13.4.2.2 Approximation (ProblemDescDB & problem_db, size_t num_vars)

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().

13.4.2.3 Approximation (const String & approx_type, const UShortArray & approx_order, size_t num_vars, short data_order, short output_level)

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().
13.4 Approximation Class Reference

13.4.2.4 Approximation (const Approximation & approx)


13.4.2.5 ~Approximation () [virtual]

Destructor. Destructor decrements referenceCount and only deletes approxRep when referenceCount reaches zero. References Approximation::approxRep, and Approximation::referenceCount.

13.4.2.6 Approximation (BaseConstructor, const 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_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). References Approximation::approxType, ProblemDescDB::get_bool(), and Dakota::strends().

13.4.2.7 Approximation (NoDBBaseConstructor, 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_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).

13.4.3 Member Function Documentation

13.4.3.1 Approximation operator= (const Approximation & approx)


13.4.3.2 void build () [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 GaussProcApproximation, PecosApproximation, SurfpackApproximation, TANA3Approximation, and TaylorApproximation.

References Dakota::abort_handler(), Approximation::approxData, Approximation::approxRep, Approximation::build(), Approximation::min_points(), and Approximation::numVars.

Referenced by Approximation::build(), and Approximation::rebuild().

13.4.3.3 void rebuild () [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().

Referenced by Approximation::rebuild().

13.4.3.4 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 Dakota::abort_handler(), Approximation::approxData, Approximation::approxRep, Approximation::pop(), and Approximation::popCountStack.

Referenced by Approximation::pop().

13.4.3.5 void restore () [virtual]

restores state prior to previous append() 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::popCountStack, Approximation::restoration_index(), and Approximation::restore().

Referenced by Approximation::restore().

13.4.3.6 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, Approximation::clear_saved(), Approximation::finalization_index(), and Approximation::finalize().
13.4 Approximation Class Reference

Referenced by Approximation::finalize().

13.4.3.7 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_current().

13.4.3.8 void clear_all() [inline]
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().

13.4.3.9 Approximation * get_approx (ProblemDescDB & problem_db, size_t num_vars) [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 ProblemDescDB::get_string(), and Dakota::strends(). Referenced by Approximation::Approximation().

13.4.3.10 Approximation * get_approx (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 approxRep to the appropriate derived type. Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
References Dakota::strends().

13.4.4 Member Data Documentation

13.4.4.1 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 SurfpackApproximation::add_sd_to_surfdata(), TaylorApproximation::build(), TANA3Approximation::build(), GaussProcApproximation::GaussProcApproximation(), TaylorApproximation::gradient(), TaylorApproximation::hessian(), TaylorApproximation::min_coefficients(), TaylorApproximation::min_points(), PecosApproximation::PecosApproximation(), Approximation::recommended_points(), SurfpackApproximation::SurfpackApproximation(), SurfpackApproximation::surrogates_to_surfdata(), TANA3Approximation::TANA3Approximation(), TaylorApproximation::TaylorApproximation(), and TaylorApproximation::value().

The documentation for this class was generated from the following files:

- DakotaApproximation.hpp
- DakotaApproximation.cpp
13.5 **ApproximationInterface Class Reference**

Derived class within the interface class hierarchy for supporting approximations to simulation-based results. Inheritance diagram for ApproximationInterface:

```
  Interface
    ApproximationInterface
```

**Public Member Functions**

- **ApproximationInterface** (ProblemDescDB &problem_db, const Variables &am_vars, bool am_cache, const String &am_interface_id, size_t num_fns)
  
  *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 functionSurfaces*

- 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_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)
- `void rebuild_approximation` (const BoolDeque &rebuild_deque)
- `void pop_approximation` (bool save_surr_data)
- `void restore_approximation` ()
- `bool restore_available` ()
- `void finalize_approximation` ()
  finalizes the approximation by applying all trial increments
- `void store_approximation` ()
  move the current approximation into storage for later combination
- `void combine_approximation` (short corr_type)
  combine the current approximation with one previously stored
- `void clear_current` ()
  clears current data from an approximation interface
- `void clear_all` ()
  clears all data from an approximation interface
- `void clear_saved` ()
  clears saved data (from pop invocations) from an approximation interface
- `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` ()
  retrieve the approximation coefficients from each Approximation within an ApproximationInterface
- `void approximation_coefficients` (const RealVectorArray &approx_coeffs)
  set the approximation coefficients within each Approximation within an ApproximationInterface
- `const RealVector & approximation_variances` (const Variables &vars)
  retrieve the approximation variances from each Approximation within an ApproximationInterface
- `const IntResponseMap & synch` ()
  recovers data from a series of asynchronous evaluations (blocking)
- `const IntResponseMap & synch_nowait` ()
  recovers data from a series of asynchronous evaluations (nonblocking)
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 update_pop_counts (const IntResponseMap &resp_map)
  append to the popCountStack within each of the functionSurfaces based on the active set definitions within resp_map

- void read_challenge_points ()
  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

- std::vector< Approximation > functionSurfaces
  list of approximations, one per response function

- RealVectorArray functionSurfaceCoeffs
  array of approximation coefficient vectors, one vector 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)

- bool challengeAnnotated
  whether the points file is annotated

- RealMatrix challengePoints
  container for the challenge points data

- 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 actualModelInterfaceId
  the interface id from the actualModel used for ordered PRPCache lookups

- IntResponseMap beforeSynchResponseMap
  bookkeeping map to catalogue responses generated in map() for use in synch() and synch_nowait(). This supports pseudo-asynchronous operations (approximate responses are always computed synchronously, but asynchronous virtual functions are supported through bookkeeping).

### 13.5.1 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.

### 13.5.2 Member Function Documentation

#### 13.5.2.1 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().

#### 13.5.2.2 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::approxFnIndices, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

#### 13.5.2.3 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::approxFnIndices, Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), and ApproximationInterface::shallow_add().

13.5.2.4 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().

13.5.2.5 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::approxFnIndices, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

13.5.2.6 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().

13.5.2.7 void build_approximation (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) [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, ApproximationInterface::challengeFile, ApproximationInterface::challengePoints, ApproximationInterface::functionSurfaces, and ApproximationInterface::readChallengePoints().

13.5.2.8 void rebuild_approximation (const BoolDeque & rebuild_deque) [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, and ApproximationInterface::functionSurfaces.

13.5.2.9 void pop_approximation (bool save_surr_data) [protected, virtual]

This function removes data provided by a previous call to append_approximation(), possibly different numbers for each function, or as specified in pop_count, which is assumed same for all functions.
Reimplemented from Interface.
References ApproximationInterface::approxFnIndices, and ApproximationInterface::functionSurfaces.

13.5.2.10 void restore_approximation () [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, and ApproximationInterface::functionSurfaces.

13.5.2.11 bool restore_available () [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, and ApproximationInterface::functionSurfaces.

13.5.3 Member Data Documentation

13.5.3.1 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_
13.5 ApproximationInterface Class Reference

variances(), ApproximationInterface::ApproximationInterface(), ApproximationInterface::approximations(), ApproximationInterface::build_approximation(), ApproximationInterface::clear_all(), ApproximationInterface::clear_current(), ApproximationInterface::clear_saved(), ApproximationInterface::combine_approximation(), ApproximationInterface::finalize_approximation(), ApproximationInterface::map(), ApproximationInterface::minimum_points(), ApproximationInterface::mixed_add(), ApproximationInterface::pop_approximation(), ApproximationInterface::read_challenge_points(), ApproximationInterface::rebuild_approximation(), ApproximationInterface::recommended_points(), ApproximationInterface::restore_approximation(), ApproximationInterface::restore_available(), 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
13.6 **APPSEvalMgr Class Reference**

Evaluation manager class for APPSPACK.

**Public Member Functions**

- **APPSEvalMgr (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::EvalRequestType 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*

- **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*

- **void set_constraint_map (std::vector<int> constraintMapIndices, std::vector<double> constraintMapMultipliers, std::vector<double> constraintMapOffsets)**
  *publishes constraint transformation*
Private Attributes

- **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

- **std::vector<int> constrMapIndices**
  map from Dakota constraint number to APPS constraint number

- **std::vector<double> constrMapMultipliers**
  multipliers for constraint transformations

- **std::vector<double> constrMapOffsets**
  offsets for constraint transformations

- **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()

### 13.6.1 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.
13.6.2 Constructor & Destructor Documentation

13.6.2.1 APPSEvalMgr (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.

13.6.3 Member Function Documentation

13.6.3.1 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.

13.6.3.2 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::asynch_compute_response(), Model::compute_response(), Model::continuous_variables(), Model::current_response(), Model::evaluation_id(), Response::function_values(), APPSEvalMgr::functionList, APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, APPSEvalMgr::numWorkersTotal, APPSEvalMgr::numWorkersUsed, APPSEvalMgr::tagList, and APPSEvalMgr::xTrial.

13.6.3.3 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::constrMapIndices, APPSEvalMgr::constrMapMultipliers, APPSEvalMgr::constrMapOffsets, APPSEvalMgr::dakotaResponseMap, APPSEvalMgr::functionList, APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, Model::num_nonlinear_eq_constraints(), APPSEvalMgr::numWorkersUsed, Model::primary_response_fn_sense(), Model::synchronize(), Model::synchronize_nowait(), and APPSEvalMgr::tagList.

The documentation for this class was generated from the following files:

- APPSEvalMgr.hpp
- APPSEvalMgr.cpp
13.7 APPSOptimizer Class Reference

Wrapper class for APPSPACK. Inheritance diagram for APPSOptimizer::

```
APPSOptimizer
  └── Optimizer
      ├── Minimizer
      └── Iterator
```

Public Member Functions

- **APPSOptimizer (Model &model)**  
  *constructor*

- **APPSOptimizer (NoDBBaseConstructor, Model &model)**  
  *alternate constructor for on-the-fly instantiations*

- **~APPSOptimizer ()**  
  *destructor*

- **void find_optimum ()**  
  *Performs the iterations to determine the optimal solution.*

Protected Member Functions

- **void set_apps_parameters ()**  
  *sets options for specific methods based on user specifications*

- **void initialize_variables_and_constraints ()**  
  *initializes problem variables and constraints*

Protected Attributes

- **HOPSPACK::ParameterList params**  
  *Pointer to APPS parameter list.*

- **HOPSPACK::ParameterList * problemParams**
• HOPSPACK::ParameterList * linearParams  
  Pointer to APPS linear constraint parameter sublist.

• HOPSPACK::ParameterList * mediatorParams  
  Pointer to APPS mediator parameter sublist.

• HOPSPACK::ParameterList * citizenParams  
  Pointer to APPS citizen/algorithm parameter sublist.

• APPSEvalMgr * evalMgr  
  Pointer to the APPS evaluation manager object.

• 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

13.7.1 Detailed Description

Wrapper class for APPSPACK. The APPSOptimizer class provides a wrapper for APPSPACK, a Sandia-developed C++ library for generalized pattern search. APPSPACK 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 APPS’s "Debug", "Maximum Evaluations", "Bounds Tolerance"/"Machine Epsilon"/"Constraint Tolerance", "Initial Step", "Contraction Factor", "Step Tolerance", "Function Tolerance", "Synchronous", "Method", "Initial Penalty Value", and "Initial Smoothing Value" data attributes. Refer to the APPS web site (http://software.sandia.gov/appspack) for additional information on APPS objects and controls.

13.7.2 Constructor & Destructor Documentation

13.7.2.1 APPSOptimizer (Model & model)

constructor Wrapper class for HOPSPACK.

The APPSOptimizer class provides a wrapper for HOPSPACK, a Sandia-developed C++ library for generalized 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.

References APPSOptimizer::evalMgr, Model::init_communicators(), Iterator::iteratedModel, Iterator::maxConcurrency, Minimizer::minimizerRecasts, and APPSOptimizer::set_apps_parameters().

13.7.3 Member Function Documentation

13.7.3.1 void find_optimum () [virtual]

Performs the iterations to determine the optimal solution. find_optimum redefines the Optimizer virtual function to perform the optimization using HOPS. It first sets up the problem data, then executes minimize() on the HOPS optimizer, and finally catalogues the results.

Implements Optimizer.

References Model::asynch_flag(), Iterator::bestResponseArray, Iterator::bestVariablesArray, APPSOptimizer::constraintMapIndices, APPSOptimizer::constraintMapMultipliers, APPSOptimizer::constraintMapOffsets, APPSOptimizer::evalMgr, Model::evaluation_capacity(), APPSOptimizer::initialize_variables_and_constraints(), Iterator::iteratedModel, Minimizer::localObjectiveRecast, Iterator::numContinuousVars, Iterator::numFunctions, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, APPSOptimizer::params, Model::primary_response_fn_sense(), APPSEvalMgr::set_asynch_flag(), and APPSEvalMgr::set_total_workers().

13.7.3.2 void set_apps_parameters () [protected]

sets options for specific methods based on user specifications Set all of the HOPS algorithmic parameters as specified in the DAKOTA input deck. This is called at construction time.

References APPSOptimizer::citizenParams, Minimizer::constraintTol, APPSOptimizer::evalMgr, ProblemDescDB::get_real(), ProblemDescDB::get_string(), ProblemDescDB::is_null(), APPSOptimizer::linearParams, Iterator::maxConcurrency, Iterator::maxFunctionEvals, APPSOptimizer::mediatorParams, Iterator::numContinuousVars, Minimizer::numNonlinearConstraints, Iterator::outputLevel, APPSOptimizer::params, Iterator::probDescDB, APPSOptimizer::problemParams, and APPSEvalMgr::set_blocking_synch().

Referenced by APPSOptimizer::APPSOptimizer().

Dakota Version 5.4 Developers Manual generated on November 7, 2013
13.7.3.3  void initialize_variables_and_constraints ()  [protected]

initializes problem variables and constraints Set the variables and constraints as specified in the DAKOTA input
deck. This is done at run time.

References Minimizer::bigRealBoundSize, APPSOptimizer::constraintMapIndices, APPSOpti-
mizer::constraintMapMultipliers, APPSOptimizer::constraintMapOffsets, Model::continuous_lower_-
bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), APPSOptimizer::evalMgr,
Iterator::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(), APPSOptimizer::linearParams, Model::nonlinear_eq_constraint_targets(),
Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Iter-
ator::numContinuousVars, Minimizer::numLinearEqConstraints, Minimizer::numLinearIneqConstraints,
Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, APPSOpti-
mizer::problemParams, and APPSEvalMgr::set_constraint_map().

Referenced by APPSOptimizer::find_optimum().

The documentation for this class was generated from the following files:

- APPSOptimizer.hpp
- APPSOptimizer.cpp
13.8 BaseConstructor Struct Reference

Dummy struct for overloading letter-envelope constructors.

Public Member Functions

- **BaseConstructor (int=0)**
  
  C++ structs can have constructors.

13.8.1 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
13.9 BiStream Class Reference

The binary input stream class. Overloads the >> operator for all data types.

Public Member Functions

- **BiStream ()**
  Default constructor, need to open.

- **BiStream (const char *s)**
  Constructor takes name of input file.

- **BiStream (const char *s, std::ios_base::openmode mode)**
  Constructor takes name of input file, mode.

- **~BiStream ()**
  Destructor, calls xdr_destroy to delete xdr stream.

- **BiStream & operator>>(std::string &ds)**
  Binary Input stream operator>>.

- **BiStream & operator>>(char &c)**
  Input operator, reads char from binary stream BiStream.

- **BiStream & operator>>(int &i)**
  Input operator, reads int* from binary stream BiStream.

- **BiStream & operator>>(long &l)**
  Input operator, reads long from binary stream BiStream.

- **BiStream & operator>>(short &s)**
  Input operator, reads short from binary stream BiStream.

- **BiStream & operator>>(bool &b)**
  Input operator, reads bool from binary stream BiStream.

- **BiStream & operator>>(double &d)**
  Input operator, reads double from binary stream BiStream.

- **BiStream & operator>>(float &f)**
  Input operator, reads float from binary stream BiStream.

- **BiStream & operator>>(unsigned char &c)**
  Input operator, reads unsigned char from binary stream BiStream.
13.9 BiStream Class Reference

- **BiStream & operator>> (unsigned int &i)**
  
  Input operator, reads unsigned int from binary stream BiStream.

- **BiStream & operator>> (unsigned long &l)**
  
  Input operator, reads unsigned long from binary stream BiStream.

- **BiStream & operator>> (unsigned short &s)**
  
  Input operator, reads unsigned short from binary stream BiStream.

**Private Attributes**

- XDR xdrInBuf
  
  XDR input stream buffer.

- char inBuf [MAX_NETOBJ_SZ]
  
  Buffer to hold data as it is read in.

13.9.1 Detailed Description

The binary input stream class. Overloads the >> operator for all data types. The Dakota::BiStream class is a binary input class which overloads the >> operator for all standard data types (int, char, float, etc). The class relies on the methods within the ifstream base class. The Dakota::BiStream class inherits from the ifstream class. If available, the class utilize rpc/xdr to construct machine independent binary files. These Dakota restart files can be moved from host to host. The motivation to develop these classes was to replace the Rogue wave classes which Dakota historically used for binary I/O.

13.9.2 Constructor & Destructor Documentation

13.9.2.1 BiStream ()

Default constructor, need to open. Default constructor, allocates xdr stream, but does not call the open method. The open method must be called before stream can be read.

References BiStream::inBuf, and BiStream::xdrInBuf.

13.9.2.2 BiStream (const char * s)

Constructor takes name of input file. Constructor which takes a char* filename. Calls the base class open method with the filename and no other arguments. Also allocates the xdr stream.

References BiStream::inBuf, and BiStream::xdrInBuf.
13.9.2.3 BiStream (const char ∗ s, std::ios_base::openmode mode)

Constructor takes name of input file, mode. Constructor which takes a char* filename and int flags. Calls the base class open method with the filename and flags as arguments. Also allocates xdr stream.

References BiStream::inBuf, and BiStream::xdrInBuf.

13.9.2.4 ~BiStream ()

Destructor, calls xdr_destroy to delete xdr stream. Destructor, destroys the xdr stream allocated in constructor

References BiStream::xdrInBuf.

13.9.3 Member Function Documentation

13.9.3.1 BiStream & operator>>(std::string & ds)

Binary Input stream operator>>. The std::string input operator must first read both the xdr buffer size and the size of the string written. Once these are read it can then read and convert the std::string correctly.

References BiStream::inBuf, and BiStream::xdrInBuf.

The documentation for this class was generated from the following files:

- DakotaBinStream.hpp
- DakotaBinStream.cpp
13.10 BoStream Class Reference

The binary output stream class. Overloads the \texttt{\langle\langle} operator for all data types.

Public Member Functions

- **BoStream ()**
  
  Default constructor, need to open.

- **BoStream (const char \*s)**
  
  Constructor takes name of input file.

- **BoStream (const char \*s, std::ios_base::openmode mode)**
  
  Constructor takes name of input file, mode.

- **\texttt{\sim}BoStream () throw ()**
  
  Destructor, calls \texttt{xdr\_destroy} to delete \texttt{xdr} stream.

- **BoStream & operator\langle\langle (const std::string &ds)**
  
  Binary Output stream operator\langle\langle.

- **BoStream & operator\langle\langle (const char &c)**
  
  Output operator, writes char to binary stream \texttt{BoStream}.

- **BoStream & operator\langle\langle (const int &i)**
  
  Output operator, writes int to binary stream \texttt{BoStream}.

- **BoStream & operator\langle\langle (const long &l)**
  
  Output operator, writes long to binary stream \texttt{BoStream}.

- **BoStream & operator\langle\langle (const short &s)**
  
  Output operator, writes short to binary stream \texttt{BoStream}.

- **BoStream & operator\langle\langle (const bool &b)**
  
  Output operator, writes bool to binary stream \texttt{BoStream}.

- **BoStream & operator\langle\langle (const double &d)**
  
  Output operator, writes double to binary stream \texttt{BoStream}.

- **BoStream & operator\langle\langle (const float &f)**
  
  Output operator, writes float to binary stream \texttt{BoStream}.

- **BoStream & operator\langle\langle (const unsigned char &c)**
  
  Output operator, writes unsigned char to binary stream \texttt{BoStream}. 
• **BoStream** & operator\textless\textless (const unsigned int &i)

  Output operator, writes unsigned int to binary stream **BoStream**.

• **BoStream** & operator\textless\textless (const unsigned long &l)

  Output operator, writes unsigned long to binary stream **BoStream**.

• **BoStream** & operator\textless\textless (const unsigned short &s)

  Output operator, writes unsigned short to binary stream **BoStream**.

**Private Attributes**

• XDR xdrOutBuf

  XDR output stream buffer.

• char outBuf [MAX_NETOBJ_SZ]

  Buffer to hold converted data before it is written.

### 13.10.1 Detailed Description

The binary output stream class. Overloads the \textless\textless operator for all data types. The **Dakota::BoStream** class is a binary output classes which overloads the \textless\textless operator for all standard data types (int, char, float, etc). The class relies on the built in write methods within the ostream base classes. **Dakota::BoStream** inherits from the ofstream class. The motivation to develop this class was to replace the Rogue wave class which **Dakota** historically used for binary I/O. If available, the class utilize rpc/xdr to construct machine independent binary files. These **Dakota** restart files can be moved between hosts.

### 13.10.2 Constructor & Destructor Documentation

#### 13.10.2.1 BoStream ()

Default constructor, need to open. Default constructor allocates the xdr stream but does not call the open() method. The open() method must be called before stream can be written to.

References **BoStream::outBuf**, and **BoStream::xdrOutBuf**.

#### 13.10.2.2 BoStream (const char * s)

Constructor takes name of input file. Constructor, takes char * filename as argument. Calls base class open method with filename and no other arguments. Also allocates xdr stream

References **BoStream::outBuf**, and **BoStream::xdrOutBuf**.
13.10.2.3 **BoStream (const char * s, std::ios_base::openmode mode)**

Constructor takes name of input file, mode. Constructor, takes char * filename and int flags as arguments. Calls base class open method with filename and flags as arguments. Also allocates xdr stream. Note: If no rpc/xdr support xdr calls are #ifdef’d out.

References BoStream::outBuf, and BoStream::xdrOutBuf.

13.10.3 **Member Function Documentation**

13.10.3.1 **BoStream & operator<< (const std::string & ds)**

Binary Output stream operator<<. The std::string operator<< must first write the xdr buffer size and the original string size to the stream. The input operator needs this information to be able to correctly read and convert the std::string.

References BoStream::outBuf, and BoStream::xdrOutBuf.

The documentation for this class was generated from the following files:

- DakotaBinStream.hpp
- DakotaBinStream.cpp
13.11  COLINApplication Class Reference

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 asynch evaluation.*

- **IntResponseMap dakota_responses**
  
  *eval_id to response mapping to cache completed jobs.*

13.11.1 Detailed Description

**COLINApplication** 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**.

13.11.2 Member Function Documentation

13.11.2.1 **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 COLINOptimizer constructors.

References **Response::active_set**, **COLINApplication::activeSet**, **Model::continuous_lower_bounds**, **Model::continuous_upper_bounds**, **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::div**, **Model::drv**, **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::nonlinear_eq_constraint_coeffs**, **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**, and **Model::primary_response_fn_sense**.

Referenced by **COLINApplication::COLINApplication**.
13.11.2.2 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 Model::asynch_compute_response(), COLINApplication::colin_request_to_dakota_request(), Model::evaluation_id(), and COLINApplication::iteratedModel.

13.11.2.3 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().

13.11.2.4 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 async evals. The domain point is guaranteed to be compatible with data type specified by map_domain(...) References COLINApplication::colin_request_to_dakota_request(), Model::compute_response(), Model::current_response(), COLINApplication::dakota_response_to_colin_response(), and COLINApplication::iteratedModel.

13.11.2.5 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.

13.11.2.6 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::div(), Model::drv(), COLINApplication::iteratedModel, Model::num_functions(), and Dakota::set_index_to_value().

Referenced by COLINApplication::perform_evaluation_impl(), and COLINApplication::spawn_evaluation_impl().

13.11.2.7 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().

13.11.2.8 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
13.12 COLINOptimizer Class Reference

Wrapper class for optimizers defined using COLIN. Inheritance diagram for COLINOptimizer:

```
     Iterator
       |
       v
    Minimizer

    Optimizer

    COLINOptimizer
```

Public Member Functions

- **COLINOptimizer (Model &model)**
  constructor

- **COLINOptimizer (const String &method_name, Model &model, int seed, int max_iter, int max_eval)**
  alternate constructor for on-the-fly instantiations

- **COLINOptimizer (const String &method_name, Model &model)**
  alternate constructor for Iterator instantiations by name

- **~COLINOptimizer ()**
  destructor

- **void reset ()**
  clears internal optimizer state

- **void find_optimum ()**
  iterates the COLIN solver to determine the optimal solution

- **bool returns_multiple_points () const**
  some COLIN methods can return multiple points

Protected Member Functions

- **void solver_setup (const String &method_name, Model &model)**
  convenience function for setting up the particular COLIN solver and appropriate Application

- **void set_rng (int seed)**
sets up the random number generator for stochastic methods

- void set_solver_parameters ()
  
  sets construct-time options for specific methods based on user specifications, including calling method-specific set functions

- void post_run (std::ostream &s)
  
  Gets the final set of points from the solver. Look up responses and sort, first according to constraint violation, then according to function value.

- std::pair<bool, bool> colin_cache_lookup (const colin::AppResponse &colinResponse, Response &tmpResponseHolder)
  
  Retrieve response from Colin AppResponse, return pair indicating success for \langle objective, constraints \rangle.

- double constraint_violation (const Response &tmpResponseHolder)
  
  Compute constraint violation, based on nonlinear constraints in iteratedModel and provided Response data.

### Protected Attributes

- short solverType
  
  COLIN solver sub-type as enumerated in COLINOptimizer.cpp.

- colin::SolverHandle colinSolver
  
  handle to the COLIN solver

- std::pair< colin::ApplicationHandle, COLINApplication * > colinProblem
  
  handle and pointer to the COLINApplication object

- colin::EvaluationManager_Base * colinEvalMgr
  
  pointer to the COLIN evalutaion manager object

- utilib::RNG * rng
  
  random number generator pointer

- bool blockingSynch
  
  the synchronization setting: true if blocking, false if nonblocking

- Real constraint_penalty
  
  Buffer to hold problem constraint_penalty parameter.

- bool constant_penalty
  
  Buffer to hold problem constant_penalty parameter.
13.12.1 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.

13.12.2 Constructor & Destructor Documentation

13.12.2.1 COLINOptimizer (Model & model)

constructor Default constructor.

References ProblemDescDB::get_int(), ProblemDescDB::get_string(), Model::init_communicators(), Iterator::iteratedModel, Iterator::maxConcurrency, Minimizer::minimizerRecasts, Iterator::probDescDB, COLINOptimizer::set_rng(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

13.12.2.2 COLINOptimizer (const String & method_name, Model & model, int seed, int max_iter, int max_eval)

alternate constructor for on-the-fly instantiations Alternate constructor for on-the-fly instantiations.

References Iterator::maxFunctionEvals, Iterator::maxIterations, COLINOptimizer::set_rng(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

13.12.2.3 COLINOptimizer (const String & method_name, Model & model)

alternate constructor for Iterator instantiations by name Alternate constructor for Iterator instantiations by name.

References COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

13.12.3 Member Function Documentation

13.12.3.1 void find_optimum () [virtual]

iterates the COLIN solver to determine the optimal solution find_optimum redefines the Optimizer virtual function to perform the optimization using COLIN. It first sets up the problem data, then executes optimize() on the COLIN solver and finallycatalogues the results.

Implements Optimizer.
References Dakota::_NPOS, Dakota::abort_handler(), Model::asynch_flag(), COLINOptimizer::blockingSynch, COLINOptimizer::colinEvalMgr, COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, Model::continuous_variables(), Model::discrete_int_sets(), Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::evaluation_capacity(), Iterator::iteratedModel, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, Iterator::outputLevel, Dakota::set_value_to_index(), and COLINOptimizer::solverType.

13.12.3.2  bool returns_multiple_points () const  [virtual]

some COLIN methods can return multiple points Designate which solvers can return multiple final points. Reimplemented from Iterator.
References COLINOptimizer::solverType.

13.12.3.3  void solver_setup (const String & method_name, Model & model)  [protected]

convenience function for setting up the particular COLIN solver and appropriate Application This convenience function is called by the constructors in order to instantiate the solver.
References COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, ProblemDescDB::get_string(), Iterator::probDescDB, and COLINOptimizer::solverType.
Referenced by COLINOptimizer::COLINOptimizer().

13.12.3.4  void set_rng (int seed)  [protected]

sets up the random number generator for stochastic methods Instantiate random number generator (RNG).
References COLINOptimizer::colinSolver, and COLINOptimizer::rng.
Referenced by COLINOptimizer::COLINOptimizer().

13.12.3.5  void set_solver_parameters ()  [protected]

sets construct-time options for specific methods based on user specifications, including calling method-specific set functions Sets solver properties based on user specifications. Called at construction time.
References Model::asynch_flag(), COLINOptimizer::blockingSynch, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, Iterator::convergenceTol, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), ProblemDescDB::get_sa(), ProblemDescDB::get_string(), ProblemDescDB::is_null(), Iterator::iteratedModel, Iterator::maxConcurrency, Iterator::maxFunctionEvals, Iterator::maxIterations, Iterator::numContinuousVars, Iterator::outputLevel, Iterator::probDescDB, and COLINOptimizer::solverType.
Referenced by COLINOptimizer::COLINOptimizer().
13.12.3.6  void post_run (std::ostream & s)  [protected, virtual]

Get the final set of points from the solver. Look up responses and sort, first according to constraint violation, then according to function value. Supplement Optimizer::post_run to first retrieve points from the Colin cache (or possibly the Dakota DB) and rank them. When complete, this function will populate bestVariablesArray and bestResponsesArray with iterator-space data, that is, in the context of the solver, leaving any further untransformation to Optimizer.

Reimplemented from Optimizer.

References Iterator::bestResponseArray, Iterator::bestVariablesArray, COLINOptimizer::colin_cache_lookup(), COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constraint_violation(), Variables::continuous_variables(), Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Model::discrete_int_sets(), Variables::discrete_int_variable(), Variables::discrete_real_variable(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Response::function_values(), Iterator::iteratedModel, Optimizer::localObjectiveRecast, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, Iterator::numFinalSolutions, Optimizer::numObjectiveFns, Minimizer::objective(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Minimizer::resize_best_response_array(), Minimizer::resize_best_variables_array(), Dakota::set_index_to_value(), Model::subordinate_model(), and Dakota::write_data().

13.12.3.7  std::pair<bool, bool> colin_cache_lookup (const colin::AppResponse & colinResponse, Response & tmpResponseHolder)  [protected]

Retrieve response from Colin AppResponse, return pair indicating success for <objective, constraints>. Encapsulated Colin Cache response extraction, which will ultimately become the default lookup. Might want to return separate vectors of function values and constraints for use in the sort, but not for now (least change). Return true if not needed or successful lookup.

References Response::function_value(), Minimizer::numNonlinearConstraints, and Optimizer::numObjectiveFns.

Referenced by COLINOptimizer::post_run().

13.12.3.8  double constraint_violation (const Response & tmpResponseHolder)  [protected]

Compute constraint violation, based on nonlinear constraints in iteratedModel and provided Response data. BMA TODO: incorporate constraint tolerance, possibly via elevating SurrBasedMinimizer::constraint_violation(). Always use iteratedModel to get the constraints; they are in the right space.

References Response::function_values(), Iterator::iteratedModel, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), and Minimizer::numIterPrimaryFns.

Referenced by COLINOptimizer::post_run().

The documentation for this class was generated from the following files:

- COLINOptimizer.hpp
- COLINOptimizer.cpp
13.13 CollaborativeHybridStrategy Class Reference

*Strategy* for hybrid minimization using multiple collaborating optimization and nonlinear least squares methods. Inheritance diagram for CollaborativeHybridStrategy:

| Strategy | HybridStrategy | CollaborativeHybridStrategy |

**Public Member Functions**

- `CollaborativeHybridStrategy (ProblemDescDB &problem_db)`
  - *constructor*
- `~CollaborativeHybridStrategy ()`
  - *destructor*

**Protected Member Functions**

- `void run_strategy ()`
  - *Performs the collaborative hybrid minimization strategy.*
- `const Variables & variables_results () const`
  - *return the final solution from the collaborative minimization (variables)*
- `const Response & response_results () const`
  - *return the final solution from the collaborative minimization (response)*

**Private Attributes**

- `String hybridCollabType`
  - *abo or hops*
- `Variables bestVariables`
  - *best variables found in minimization*
- `Response bestResponse`
  - *best response found in minimization*
13.13.1 Detailed Description

Strategy for hybrid minimization using multiple collaborating optimization and nonlinear least squares methods. This strategy has two approaches to hybrid minimization: (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:

- CollaborativeHybridStrategy.hpp
- CollaborativeHybridStrategy.cpp
13.14 CommandLineHandler Class Reference

Utility class for managing command line inputs to DAKOTA. Inheritance diagram for CommandLineHandler::

```
CommandLineHandler
     |                     GetLongOpt
    |                       |
    V
 CommandLineHandler
```

Public Member Functions

- **CommandLineHandler ()**
  *default constructor, requires check_usage() call for parsing*

- **CommandLineHandler (int argc, char **argv)**
  *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*.

- **bool instantiate_flag () const**
  *Whether command line args dictate instantiation of objects for run.*

- **bool run_flag () const**
  *Whether steps beyond check are requested.*

Private Member Functions

- **void initialize_options ()**
  *enrolls the supported command line inputs.*

- **void assign_streams ()**
  *conditionally associate Cout/Cerr with file streams, if specified by user*

- **void reset_streams ()**
Class Documentation

conditionally restore Cout/Cerr to default

- void output_version () const
  outputs the DAKOTA version

- void output_helper (const std::string message, std::ostream &os) const
  perform output of message to ostream os on rank 0 only

Private Attributes

- std::ofstream output_ofstream
  temporary file redirection of stdout

- std::ofstream error_ofstream
  temporary file redirection of stderr

13.14.1 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.

13.14.2 Member Function Documentation

13.14.2.1 bool instantiate_flag () const [inline]

Whether command line args dictate instantiation of objects for run. Instantiate objects if not just getting help or version

References GetLongOpt::retrieve().

Referenced by main().

13.14.2.2 void assign_streams () [private]

conditionally associate Cout/Cerr with file streams, if specified by user Redirect output/error to files, including output from this class. If there is a valid ParallelLibrary, only redirect on rank 0 to avoid file clash.

References Dakota::abort_handler(), Dakota::Dak_pl, Dakota::dakota_cerr, Dakota::dakota_cout, CommandLineHandler::error_ofstream, CommandLineHandler::output_helper(), CommandLineHandler::output_ofstream, GetLongOpt::retrieve(), and ParallelLibrary::world_rank().

Referenced by CommandLineHandler::check_usage().
13.14.2.3 void output_version () const [private]

outputs the DAKOTA version Version is always output to Cout
References CommandLineHandler::output_helper().
Referenced by CommandLineHandler::check_usage().

13.14.2.4 void output_helper (const std::string message, std::ostream & os) const [private]

perform output of message to ostream os on rank 0 only When there is a valid ParallelLibrary, output only on rank 0
References Dakota::Dak_pl, and ParallelLibrary::output_helper().
Referenced by CommandLineHandler::assign_streams(), CommandLineHandler::check_usage(), and CommandLineHandler::output_version().

The documentation for this class was generated from the following files:

- CommandLineHandler.hpp
- CommandLineHandler.cpp
13.15 CommandShell Class Reference

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

Public Member Functions

- **CommandShell** (const std::string &work_dir)
  
  *constructor*

- **~CommandShell** ()
  
  *destructor*

- **CommandShell & operator<<** (const char *cmd)
  
  *appends cmd to sysCommand*

- **CommandShell & operator<<** (const std::string &cmd)
  
  *convenient operator: appends string to the commandString to be executed*

- **CommandShell & operator<<** (CommandShell &(*f)(CommandShell &))
  
  *allows passing of the flush function to the shell using <<*

- **CommandShell & 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

- const std::string & **workDir**
  
  *To convey working directory when useWorkdir is true:*

- std::string **sysCommand**
  
  *The command string that is constructed through one or more << insertions and then executed by flush.*
13.15 CommandShell Class Reference

- bool asynchFlag
  flags nonblocking operation (background system calls)

- bool suppressOutputFlag
  flags suppression of shell output (no command echo)

13.15.1 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.

13.15.2 Member Function Documentation

13.15.2.1 CommandShell & operator<< (const char * cmd) [inline]

appends cmd to sysCommand convenient operator: appends string to the commandString to be executed

References CommandShell::sysCommand.

13.15.2.2 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 <<

13.15.2.3 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, CommandShell::sysCommand, and CommandShell::workDir.

The documentation for this class was generated from the following files:

- CommandShell.hpp
- CommandShell.cpp
13.16 ConcurrentStrategy Class Reference

Strategy for multi-start iteration or pareto set optimization. Inheritance diagram for ConcurrentStrategy::

```
Strategy
  └── ConcurrentStrategy
```

Public Member Functions

- **ConcurrentStrategy (ProblemDescDB &problem_db)**
  
  *constructor*

- **∼ConcurrentStrategy ()**
  
  *destructor*

Protected Member Functions

- **void run_strategy ()**

  *Performs the concurrent strategy by executing selectedIterator on userDefinedModel multiple times in parallel for different settings within the iterator or model.*

- **void initialize_iterator (int job_index)**

  *initialize the iterator about to be executed within a parallel iterator scheduling function (serve_iterators() or static_schedule_iterators())*

- **void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)**

  *pack a send_buffer for assigning an iterator job to a server*

- **void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer)**

  *unpack a recv_buffer for accepting an iterator job from the scheduler*

- **void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)**

  *pack a send_buffer for returning iterator results from a server*

- **void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)**

  *unpack a recv_buffer for accepting iterator results from a server*

- **void update_local_results (int job_index)**

  *update local PRP results arrays with current iteration results*
Private Member Functions

- void initialize_iterator (const RealVector &param_set)
  
  called by unpack_parameters_buffer(MPIUnpackBuffer) and initialize_iterator(int) to update userDefinedModel
  and selectedIterator

- void print_results() const
  
  prints the concurrent iteration results summary (called by run_strategy())

Private Attributes

- Model userDefinedModel
  
  the model used by the iterator

- Iterator selectedIterator
  
  the iterator used by the concurrent strategy

- bool multiStartFlag
  
  distinguishes multi-start from Pareto-set

- RealVector initialPt
  
  the initial continuous variables for restoring the starting point in the Pareto set strategy

- RealVectorArray parameterSets
  
  an array of parameter set vectors (either multistart variable sets or pareto multi-objective/least squares weighting
  sets) to be performed.

- PRPArray prpResults
  
  1-d array of ParamResponsePair results corresponding to numIteratorJobs

13.16.1 Detailed Description

Strategy for multi-start iteration or pareto set optimization. This strategy maintains two concurrent iterator capa-

bilities. 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 map-

ping the "pareto frontier" (the set of optimal solutions in multiobjective formulations) is provided. This pareto set

is mapped through running an optimizer multiple times for different sets of multiobjective weightings.

13.16.2 Member Function Documentation

13.16.2.1 void pack_parameters_buffer (MPIPackBuffer & send_buffer, int job_index) [inline, protected, virtual]  

pack a send_buffer for assigning an iterator job to a server This virtual function redefinition is executed on the
dedicated master processor for self scheduling. It is not used for peer partitions.
Reimplemented from Strategy.
References ConcurrentStrategy::parameterSets.

13.16.2.2 void unpack_parameters_buffer (MPIUnpackBuffer & recv_buffer) [inline, protected, virtual]

unpack a recv_buffer for accepting an iterator job from the scheduler This virtual function redefinition is executed
on an iterator server for dedicated master self scheduling. It is not used for peer partitions.
Reimplemented from Strategy.
References ConcurrentStrategy::initialize_iterator().

13.16.2.3 void pack_results_buffer (MPIPackBuffer & send_buffer, int job_index) [inline, protected, virtual]

pack a send_buffer for returning iterator results from a server This virtual function redefinition is executed either
on an iterator server for dedicated master self scheduling or on peers 2 through n for static scheduling.
Reimplemented from Strategy.
References ConcurrentStrategy::prpResults.

13.16.2.4 void unpack_results_buffer (MPIUnpackBuffer & recv_buffer, int job_index) [inline, protected, virtual]

unpack a recv_buffer for accepting iterator results from a server This virtual function redefinition is executed on
an strategy master (either the dedicated master processor for self scheduling or peer 1 for static scheduling).
Reimplemented from Strategy.
References ConcurrentStrategy::prpResults.
The documentation for this class was generated from the following files:

- ConcurrentStrategy.hpp
- ConcurrentStrategy.cpp
13.17 CONMINOptimizer Class Reference

Wrapper class for the CONMIN optimization library. Inheritance diagram for CONMINOptimizer:

```
CONMINOptimizer
  |        |
  |        |
  V        V
Minimizer Optimizer
  |        |
  |        |
  V        V
Iterator
```

Public Member Functions

- `CONMINOptimizer (Model &model)`  
  *standard constructor*

- `CONMINOptimizer (NoDBBaseConstructor, Model &model)`  
  *alternate constructor*

- `~CONMINOptimizer ()`  
  *destructor*

- `void find_optimum ()`  
  *Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- `void initialize_run ()`  
  *performs run-time set up*

Private Member Functions

- `void initialize ()`  
  *Shared constructor code.*

- `void allocate_workspace ()`  
  *Allocates workspace for the optimizer.*
- void deallocate_workspace ()
  
  Releases workspace memory.

- void allocate_constraints ()
  
  Allocates constraint mappings.

**Private Attributes**

- int conminInfo
  
  INFO from CONMIN manual.

- int printControl
  
  IPRINT from CONMIN manual (controls output verbosity).

- int optimizationType
  
  MINMAX from DOT manual (minimize or maximize).

- Real objFnValue
  
  value of the objective function passed to CONMIN

- RealVector constraintValues
  
  array of nonlinear constraint values passed to CONMIN

- int numConminNlnConstr
  
  total number of nonlinear constraints seen by CONMIN

- int numConminLinConstr
  
  total number of linear constraints seen by CONMIN

- int numConminConstr
  
  total number of linear and nonlinear constraints seen by CONMIN

- SizetArray constraintMappingIndices
  
  a container of indices for referencing the corresponding Response constraints used in computing the CONMIN constraints.

- RealArray constraintMappingMultipliers
  
  a container of multipliers for mapping the Response constraints to the CONMIN constraints.

- RealArray constraintMappingOffsets
  
  a container of offsets for mapping the Response constraints to the CONMIN constraints.

- int N1
  
  Size variable for CONMIN arrays. See CONMIN manual.
• int N2
  Size variable for CONMIN arrays. See CONMIN manual.

• int N3
  Size variable for CONMIN arrays. See CONMIN manual.

• int N4
  Size variable for CONMIN arrays. See CONMIN manual.

• int N5
  Size variable for CONMIN arrays. See CONMIN manual.

• int NFDG
  Finite difference flag.

• int IPRINT
  Flag to control amount of output data.

• int ITMAX
  Flag to specify the maximum number of iterations.

• double FDCH
  Relative finite difference step size.

• double FDCHM
  Absolute finite difference step size.

• double CT
  Constraint thickness parameter.

• double CTMIN
  Minimum absolute value of CT used during optimization.

• double CTL
  Constraint thickness parameter for linear and side constraints.

• double CTLMIN
  Minimum value of CTL used during optimization.

• double DELFUN
  Relative convergence criterion threshold.

• double DABFUN
  Absolute convergence criterion threshold.

• double * conminDesVars
Array of design variables used by CONMIN (length $NI = numdv+2$).

- double * conminLowerBnds
  
  Array of lower bounds used by CONMIN (length $NI = numdv+2$).

- double * conminUpperBnds
  
  Array of upper bounds used by CONMIN (length $NI = numdv+2$).

- double * S
  
  Internal CONMIN array.

- double * G1
  
  Internal CONMIN array.

- double * G2
  
  Internal CONMIN array.

- double * B
  
  Internal CONMIN array.

- double * C
  
  Internal CONMIN array.

- int * MS1
  
  Internal CONMIN array.

- double * SCAL
  
  Internal CONMIN array.

- double * DF
  
  Internal CONMIN array.

- double * A
  
  Internal CONMIN array.

- int * ISC
  
  Internal CONMIN array.

- int * IC
  
  Internal CONMIN array.
13.17 CONMINOptimizer Class Reference

13.17.1 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 NPSOLOptimizer 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 find_optimum() loop since there is no CON-
MIN 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 NFDG 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.

13.17.2 Member Data Documentation

13.17.2.1 int conminInfo  [private]

INFO from CONMIN manual. Information requested by CONMIN: 1 = evaluate objective and constraints, 2 =
evaluate gradients of objective and constraints.

Referenced by CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize().

13.17.2.2 int printControl  [private]

IPRINT from CONMIN manual (controls output verbosity). Values range from 0 (nothing) to 4 (most output).
0 = nothing, 1 = initial and final function information, 2 = all of #1 plus function value and design vars at each
iteration, 3 = all of #2 plus constraint values and direction vectors, 4 = all of #3 plus gradients of the objective
function and constraints, 5 = all of #4 plus proposed design vector, plus objective and constraint functions from
the 1-D search

Referenced by CONMINOptimizer::initialize().

13.17.2.3 int optimizationType  [private]

MINMAX from DOT manual (minimize or maximize). Values of 0 or -1 (minimize) or 1 (maximize).

13.17.2.4 RealVector constraintValues  [private]

array of nonlinear constraint values passed to CONMIN This array must be of nonzero length and must contain
only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities
and equalities).

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().
13.17.2.5 SizetArray constraintMappingIndices [private]

A container of indices for referencing the corresponding Response constraints used in computing the CONMIN constraints. The length of the container corresponds to the number of CONMIN constraints, and each entry in the container points to the corresponding DAKOTA constraint.

Referenced by CONMINOptimizer::allocate_constraints(), and CONMINOptimizer::find_optimum().

13.17.2.6 RealArray constraintMappingMultipliers [private]

A container of multipliers for mapping the Response constraints to the CONMIN constraints. The length of the container corresponds to the number of CONMIN constraints, and each entry in the container stores a multiplier for the DAKOTA constraint identified with constraintMappingIndices. These multipliers are currently +1 or -1.

Referenced by CONMINOptimizer::allocate_constraints(), and CONMINOptimizer::find_optimum().

13.17.2.7 RealArray constraintMappingOffsets [private]

A container of offsets for mapping the Response constraints to the CONMIN constraints. The length of the container corresponds to the number of CONMIN constraints, and each entry in the container stores an offset for the DAKOTA constraint identified with constraintMappingIndices. These offsets involve inequality bounds or equality targets, since CONMIN assumes constraint allowables = 0.

Referenced by CONMINOptimizer::allocate_constraints(), and CONMINOptimizer::find_optimum().

13.17.2.8 int N1 [private]

Size variable for CONMIN arrays. See CONMIN manual. N1 = number of variables + 2

Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize_run().

13.17.2.9 int N2 [private]

Size variable for CONMIN arrays. See CONMIN manual. N2 = number of constraints + 2+(number of variables)

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

13.17.2.10 int N3 [private]

Size variable for CONMIN arrays. See CONMIN manual. N3 = Maximum possible number of active constraints.

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

13.17.2.11 int N4 [private]

Size variable for CONMIN arrays. See CONMIN manual. N4 = Maximum(N3,number of variables)

Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().
13.17.2.12 int N5 [private]

Size variable for CONMIN arrays. See CONMIN manual. \( N5 = 2 \times (N4) \)
Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::find_optimum().

13.17.2.13 double CT [private]

Constraint thickness parameter. The value of CT decreases in magnitude during optimization.
Referenced by CONMINOptimizer::find_optimum(), and CONMINOptimizer::initialize().

13.17.2.14 double* S [private]

Internal CONMIN array. Move direction in N-dimensional space.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.15 double* G1 [private]

Internal CONMIN array. Temporary storage of constraint values.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.16 double* G2 [private]

Internal CONMIN array. Temporary storage of constraint values.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.17 double* B [private]

Internal CONMIN array. Temporary storage for computations involving array S.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.18 double* C [private]

Internal CONMIN array. Temporary storage for use with arrays B and S.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().
13.17.2.19 int∗ MS1 [private]

Internal CONMIN array. Temporary storage for use with arrays B and S.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.20 double∗ SCAL [private]

Internal CONMIN array. Vector of scaling parameters for design parameter values.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.21 double∗ DF [private]

Internal CONMIN array. Temporary storage for analytic gradient data.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.22 double∗ A [private]

Internal CONMIN array. Temporary 2-D array for storage of constraint gradients.
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(), and
CONMINOptimizer::find_optimum().

13.17.2.23 int∗ ISC [private]

Internal CONMIN array. Array of flags to identify linear constraints. (not used in this implementation of CONMIN)
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(),
CONMINOptimizer::find_optimum() and CONMINOptimizer::initialize_run().

13.17.2.24 int∗ IC [private]

Internal CONMIN array. Array of flags to identify active and violated constraints
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::deallocate_workspace(),
CONMINOptimizer::find_optimum() and CONMINOptimizer::initialize_run().

The documentation for this class was generated from the following files:

• CONMINOptimizer.hpp
• CONMINOptimizer.cpp
13.18 Constraints Class Reference

Base class for the variable constraints class hierarchy. Inheritance diagram for Constraints::

```
Constraints
   +-----+-----+
   |     |     |
   | 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

- virtual void **reshape (const SizetArray &vc_totals)**
  reshape the lower/upper bound arrays within the Constraints hierarchy

- **const RealVector & continuous_lower_bounds () const**
  return the active continuous variable lower bounds

- **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

• **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 & discrete_int_lower_bounds () const**  
  return the active discrete variable lower bounds

• **void discrete_int_lower_bounds (const IntVector &dil_bnds)**  
  set the active discrete variable lower bounds

• **void discrete_int_lower_bound (int dil_bnd, size_t i)**  
  set an active discrete variable lower bound

• **const IntVector & discrete_int_upper_bounds () const**  
  return the active discrete variable upper bounds

• **void discrete_int_upper_bounds (const IntVector &diu_bnds)**  
  set the active discrete variable upper bounds

• **void discrete_int_upper_bound (int diu_bnd, size_t i)**  
  set an active discrete variable upper bound

• **const RealVector & discrete_real_lower_bounds () const**  
  return the active discrete variable lower bounds

• **void discrete_real_lower_bounds (const RealVector &drl_bnds)**  
  set the active discrete variable lower bounds

• **void discrete_real_lower_bound (Real drl_bnd, size_t i)**  
  set an active discrete variable lower bound

• **const RealVector & discrete_real_upper_bounds () const**  
  return the active discrete variable upper bounds

• **void discrete_real_upper_bounds (const RealVector &dru_bnds)**  
  set the active discrete variable upper bounds

• **void discrete_real_upper_bound (Real dru_bnd, size_t i)**  
  set an active discrete variable upper bound
set an active discrete variable upper bound

- `const RealVector & inactive_continuous_lower_bounds () const`
  return the inactive continuous lower bounds

- `void inactive_continuous_lower_bounds (const RealVector &icl_bnds)`
  set the inactive continuous lower bounds

- `const RealVector & inactive_continuous_upper_bounds () const`
  return the inactive continuous upper bounds

- `void inactive_continuous_upper_bounds (const RealVector &icu_bnds)`
  set the inactive continuous upper bounds

- `const IntVector & inactive_discrete_int_lower_bounds () const`
  return the inactive discrete lower bounds

- `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
• 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_lower_bounds() const
  return the nonlinear inequality constraint lower bounds

- void nonlinear_ineq_constraint_lower_bounds(const RealVector &nlneq_l_bnds)
  set the nonlinear inequality constraint lower 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 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 SizetArray &vc_totals)
  reshape the linear/nonlinear/bound constraint arrays within the Constraints hierarchy

- 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 within the Constraints hierarchy

- 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)

- virtual void build_active_views()
  construct active views of all variables bounds arrays

- virtual void build_inactive_views()
construct inactive views of all variables bounds arrays

- void build_views()
  construct active/inactive views of all variables 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*

- 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

### 13.18.1 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.
13.18.2 Constructor & Destructor Documentation

13.18.2.1 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.

13.18.2.2 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().

13.18.2.3 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().

13.18.2.4 Constraints (const Constraints & con)

copy constructor Copy constructor manages sharing of constraintsRep and incrementing of referenceCount.

References Constraints::constraintsRep, and Constraints::referenceCount.

13.18.2.5 ~Constraints () [virtual]

destructor Destructor decrements referenceCount and only deletes constraintsRep when referenceCount reaches zero.

References Constraints::constraintsRep, and Constraints::referenceCount.

13.18.2.6 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).
13.18 Constraints Class Reference

13.18.2.7 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() 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).

13.18.3 Member Function Documentation

13.18.3.1 Constraints operator= (const Constraints & con)


References Constraints::constraintsRep, and Constraints::referenceCount.

13.18.3.2 void reshape (const SizetArray & vc_totals) [virtual]

reshape the lower/upper bound arrays within the Constraints hierarchy Resizes the derived bounds arrays.

Reimplemented in MixedVarConstraints, and RelaxedVarConstraints.

References Constraints::constraintsRep, Constraints::continuousLowerBnds, Constraints::discreteIntLowerBnds, Constraints::discreteRealLowerBnds, Constraints::linearEqConCoeffs, Constraints::linearIneqConCoeffs, Constraints::numLinearEqCons, Constraints::numLinearIneqCons, and Constraints::reshape().

Referenced by DataFitSurrModel::DataFitSurrModel(), RecastModel::RecastModel(), and Constraints::reshape().

13.18.3.3 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::nonlinearIneqConCoeffs, Constraints::nonlinearIneqConLowerBnds, Constraints::nonlinearIneqConUpperBnds, Constraints::numLinearEqCons, Constraints::numLinearIneqCons, Constraints::numNonlinearEqCons, Constraints::numNonlinearIneqCons, and Constraints::sharedVarsData.

Referenced by SurrogateModel::force_rebuild(), and RecastModel::RecastModel().
13.18.3.4 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 within the Constraints hierarchy. Resizes the linear and nonlinear constraint arrays at the base class. Does NOT currently resize the derived bounds arrays.

References Constraints::constraintsRep, Constraints::linearEqConTargets, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::nonlinearEqConTargets, Constraints::nonlinearIneqConLowerBnds, Constraints::nonlinearIneqConUpperBnds, Constraints::numLinearEqCons, Constraints::numLinearIneqCons, Constraints::numNonlinearEqCons, Constraints::numNonlinearIneqCons, and Constraints::reshape().

13.18.3.5 void build_views () [inline, protected]

construct active/inactive views of all variables arrays

= EMPTY)

References Constraints::build_active_views(), Constraints::build_inactive_views(), Constraints::sharedVarsData, and SharedVariablesData::view().

Referenced by Constraints::copy(), MixedVarConstraints::MixedVarConstraints(), RelaxedVarConstraints::RelaxedVarConstraints(), RelaxedVarConstraints::reshape(), and MixedVarConstraints::reshape().

13.18.3.6 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 MixedVarConstraints::MixedVarConstraints(), and RelaxedVarConstraints::RelaxedVarConstraints().

13.18.3.7 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().
13.18.3.8 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
13.19 DataFitSurrModel Class Reference

Derived model class within the surrogate model branch for managing data fit surrogates (global and local). Inheritance diagram for DataFitSurrModel:

```
  Model  
   |    
  SurrogateModel  
  DataFitSurrModel
```

Public Member Functions

- **DataFitSurrModel (ProblemDescDB &problem_db)**
  
  *constructor*

- **DataFitSurrModel (Iterator &dace_iterator, Model &actual_model, 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 &export_points_file=String(), bool export_annotated=true, const String &import_points_file=String(), bool import_annotated=true)**

  *alternate constructor for instantiations on the fly*

- **~DataFitSurrModel ()**

  *destructor*

- **void total_points (int points)**

  *set pointsTotal and pointsManagement mode*

Protected Member Functions

- **void derived_compute_response (const ActiveSet &set)**

  *portion of compute_response() specific to DataFitSurrModel*

- **void derived_asynch_compute_response (const ActiveSet &set)**

  *portion of asynch_compute_response() 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 (bool recurse_flag=true)
  
  pass request to actualModel if recursing and then update from it

• **Interface** & 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 ()
  
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points.

• bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)
  
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the vars/response anchor point.

• void update_approximation (bool rebuild_flag)
  
  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)
  
  replaces the anchor point, and rebuilds the approximation if requested

• void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  
  replaces the current points array and rebuilds the approximation if requested
void append_approximation (bool rebuild_flag)
  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)
  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)
  appends an array 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 restore_approximation ()
  restore a previous approximation data state

bool restore_available ()
  query for whether a trial increment is restorable

void finalize_approximation ()
  finalize data fit by applying all previous trial increments

void store_approximation ()
  store the current data fit approximation for later combination

void combine_approximation (short corr_type)
  combine the current data fit approximation with one previously stored

std::vector< Approximation > & approximations ()
  retrieve the set of Approximations from approxInterface

const RealVectorArray & approximation_coefficients ()
  return the approximation coefficients from each Approximation (request forwarded to approxInterface)

void approximation_coefficients (const RealVectorArray &approx_coeffs)
  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
- void **derived_init_communicators** (int max_iterator_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** (int max_iterator_concurrency, bool recurse_flag=true)
  
  *set active parallel configuration within actualModel*

- void **derived_free_communicators** (int max_iterator_concurrency, bool recurse_flag=true)
  
  *deallocation communicator partitions for the DataFitSurrModel (request forwarded to actualModel)*

- void **serve** (int max_iterator_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*

- int **evaluation_id** () const
  
  *return the current evaluation id for the DataFitSurrModel*

- void **set_evaluation_reference** ()
  
  *set the evaluation counter reference points for the DataFitSurrModel (request forwarded to approxInterface and actualModel)*

- void **fine_grained_evaluation_counters** ()
  
  *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)*

- virtual void **eval_tag_prefix** (const String &eval_id_str)
  
  *set the hierarchical eval ID tag prefix*
Private Member Functions

- **void import_points** (bool annotated)
  optionally read surrogate data points from provided file

- **void derived_synchronize_approx** (const IntResponseMap &approx_resp_map, 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 update_actual_model** ()
  update actualModel with data from current variables/labels/bounds/targets

- **void update_from_actual_model** ()
  update current variables/labels/bounds/targets with data from actualModel

- **bool inside** (const RealVector &c_vars, const IntVector &di_vars, const RealVector &dr_vars)
  test if c_vars and d_vars are within [c_l_bnds,c_u_bnds] and [d_l_bnds,d_u_bnds]

Private Attributes

- **int surrModelEvalCntr**
  number of calls to derived_compute_response() / derived_asynch_compute_response()

- **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_points_file specification*

- String `exportPointsFile`
  
  *file name from export_points_file specification*

- bool `exportAnnotated`
  
  *annotation setting for file export of variables and approximate responses*

- `std::ofstream` `exportFileStream`
  
  *file name for export_points_file specification*

- `VariablesList` `reuseFileVars`
  
  *array of variables sets read from the import_points_file*

- `ResponseList` `reuseFileResponses`
  
  *array of response sets read from the import_points_file*

- `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)*

- String `evalTagPrefix`
  
  *cached evalTag Prefix from parents to use at compute_response time*

### 13.19.1 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.
13.19.2 Constructor & Destructor Documentation

13.19.2.1 ~DataFitSurrModel () [inline]

destructor Virtual destructor handles referenceCount at Strategy level.
References DataFitSurrModel::exportFileStream, and DataFitSurrModel::exportPointsFile.

13.19.3 Member Function Documentation

13.19.3.1 void derived_compute_response (const ActiveSet & set) [protected, virtual]

portion of compute_response() 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 Dakota::_NPOS, DiscrepancyCorrection::active(), Response::active_set(), DataFitSurrModel::actualModel, DiscrepancyCorrection::apply(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(), DataFitSurrModel::build_approximation(), DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), Model::compute_response(), Response::copy(), Model::current_response(), Model::currentResponse, Model::currentVariables, SurrogateModel::deltaCorr, Model::eval_tag_prefix(), DataFitSurrModel::evalTagPrefix, DataFitSurrModel::exportAnnotated, DataFitSurrModel::exportFileStream, DataFitSurrModel::exportPointsFile, SurrogateModel::force_rebuild(), Model::hierarchicalTagging, Interface::map(), Model::outputLevel, ActiveSet::request_vector(), SurrogateModel::response_mapping(), SurrogateModel::responseMode, DataFitSurrModel::surrModelEvalCntr, Response::update(), DataFitSurrModel::update_actual_model(), and Dakota::write_data_tabular().

13.19.3.2 void derived_asynch_compute_response (const ActiveSet & set) [protected, virtual]

portion of asynch_compute_response() 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(), Model::asynch_compute_response(), DataFitSurrModel::build_approximation(), Variables::copy(), Model::currentResponse, Model::currentVariables, Model::eval_tag_prefix(), DataFitSurrModel::evalTagPrefix, Interface::evaluation_id(), Model::evaluation_id(), DataFitSurrModel::exportPointsFile, SurrogateModel::force_rebuild(), Model::hierarchicalTagging, Interface::map(), SurrogateModel::rawVarsMap, ActiveSet::request_vector(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, DataFitSurrModel::surrModelEvalCntr, SurrogateModel::truthIdMap, and DataFitSurrModel::update_actual_model().
13.19.3.3 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_asynch_compute_response() may be inconsistent in its use of actual evaluations, approximate evaluations, or both.

Reimplemented from Model.

References DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), SurrogateModel::deltaCorr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, SurrogateModel::response_mapping(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, Interface::synch(), Model::synchronize(), and SurrogateModel::truthIdMap.

13.19.3.4 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_asynch_compute_response() may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from Model.

References Dakota::abort_handler(), DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, SurrogateModel::cachedApproxRespMap, DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), SurrogateModel::deltaCorr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, SurrogateModel::responseMapping(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, Interface::synch_nowait(), Model::synchronize_nowait(), and SurrogateModel::truthIdMap.

13.19.3.5 void build_approximation () [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}.

Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, Interface::build_approximation(), DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), Interface::clear_current(), Model::continuous_lower_bounds(), Constraints::continuous_lower_bounds(), Model::continuous_upper_bounds(), Constraints::continuous_upper_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_lower_bounds(), Model::discrete_int_upper_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_upper_bounds(), Constraints::discrete_real_upper_bounds(), Model::is_null(), Dakota::strbegins(), Model::surrogateType, DataFitSurrModel::update_actual_model(), DataFitSurrModel::update_global(), DataFitSurrModel::update_local_multipoint(), and Model::userDefinedConstraints.

Referenced by DataFitSurrModel::derived_asynch_compute_response(), and DataFitSurrModel::derived_compute_response().

Dakota Version 5.4 Developers Manual generated on November 7, 2013
13.19.3.6 bool build_approximation (const Variables & vars, const IntResponsePair & response_pr)  
[protected, virtual]

Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the 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, DataFitSur-

13.19.3.7 void update_approximation (bool rebuild_flag)  [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(), Surrogate-

13.19.3.8 void update_approximation (const Variables & vars, const IntResponsePair & response_pr,  
bool rebuild_flag)  [protected, virtual]

replaces the anchor point, and rebuilds the approximation if requested. This function populates/replaces Surro-

gateData::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, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().
13.19.3.9  void update_approximation (const VariablesArray & vars_array, const IntResponseMap & resp_map, bool rebuild_flag)  [protected, virtual]

replaces the current points array and rebuilds the approximation if requested This function populates/replaces SurrogateModel::{vars,resp}Data and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateModel::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, Interface::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

13.19.3.10  void append_approximation (bool rebuild_flag)  [protected, virtual]

appends daceIterator results to a global approximation and rebuilds it if requested This function appends one point to SurrogateModel::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateModel::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, Interface::rebuild_approximation(), and Model::surrogateType.

13.19.3.11  void append_approximation (const Variables & vars, const IntResponsePair & response_pr, bool rebuild_flag)  [protected, virtual]

appends a point to a global approximation and rebuilds it if requested This function appends one point to SurrogateModel::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateModel::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, Interface::rebuild_approximation(), and Model::surrogateType.

13.19.3.12  void append_approximation (const VariablesArray & vars_array, const IntResponseMap & resp_map, bool rebuild_flag)  [protected, virtual]

appends an array of points to a global approximation and rebuilds it if requested This function appends multiple points to SurrogateModel::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateModel::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, DataFitSur-
model::approxInterface, Model::numFns, Interface::rebuild_approximation(), and Model::surrogateType.

13.19.3.13 void derived_init_communicators (int max_iterator_concurrency, bool recurse_flag = true) [inline, protected, virtual]

set up actualModel for parallel operations asynchronous flags need to be initialized for the sub-models. In
addition, max_iterator_concurrency is the outer level iterator concurrency, not the DACE concurrency that ac-
tualModel 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, DataFitSur-
model::daceIterator, Model::derivative_concurrency(), Model::init_communicators(), Iterator::is_null(),
Model::is_null(), Iterator::maximum_concurrency(), and Interface::minimum_points().

13.19.3.14 int evaluation_id () const [inline, protected, virtual]

return the current evaluation id for the DataFitSurrModel return the DataFitSurrModel evaluation count. Due to
possibly intermittent use of surrogate bypass, this is not the same as either the approxInterface or actualModel
model evaluation counts. It also does not distinguish duplicate evals.
Reimplemented from Model.
References DataFitSurrModel::surrModelEvalCntr.

13.19.3.15 void import_points (bool annotated) [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()
References DataFitSurrModel::actualModel, Model::continuous_variable_ids(), Variables::continuous_-
variable_ids(), Model::current_variables(), Model::currentVariables, Model::cv(), Variables::cv(),
ActiveSet::derivative_vector(), DataFitSurrModel::importPointsFile, Model::is_null(), Model::numFns,
Model::outputLevel, Dakota::read_data_tabular(), DataFitSurrModel::reuseFileResponses, DataFitSur-
model::reuseFileVars, and Variables::shared_data().
Referenced by DataFitSurrModel::DataFitSurrModel().

13.19.3.16 void build_global () [private]

builds a global approximation using daceIterator. Determine 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(), Iterator::active_set(), DataFitSurrModel::actualModel, Iterator::all_-
responses(), Iterator::all_samples(), Iterator::all_variables(), Interface::append_approximation(),
Interface::approximation_data(), DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(),
13.19 DataFitSurrModel Class Reference

Iterator::compact_mode(), DataFitSurrModel::component_parallel_mode(), Variables::continuous_variables(), Model::currentVariables, Model::cv(), Variables::cv(), DataFitSurrModel::daceIterator, Dakota::data_pairs, Variables::discrete_int_variables(), Variables::discrete_real_variables(), Model::div(), Variables::div(), Model::drv(), Variables::drv(), Iterator::eval_tag_prefix(), DataFitSurrModel::evalTagPrefix, Model::hierarchicalTagging, DataFitSurrModel::importPointsFile, DataFitSurrModel::inside(), Model::interface_id(), RecastModel::inverse_transform_variables(), Iterator::is_null(), Model::is_null(), Interface::minimum_points(), Iterator::num_samples(), Model::outputLevel, DataFitSurrModel::pointReuse, DataFitSurrModel::pointsManagement, DataFitSurrModel::pointsTotal, Interface::recommended_points(), DataFitSurrModel::reuseFileResponses, DataFitSurrModel::reuseFileVars, Iterator::run_iterator(), Iterator::sampling_reset(), Model::subordinate_models(), DataFitSurrModel::surrModelEvalCntr, and SurrogateModel::surrogatFnIndices.

Referenced by DataFitSurrModel::build_approximation().

13.19.3.17 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::actualModel, DataFitSurrModel::approxInterface, SurrogateModel::asv_mapping(), DataFitSurrModel::component_parallel_mode(), Model::compute_response(), Model::continuous_variable_ids(), Model::current_response(), Model::current_variables(), Model::evaluation_id(), Model::hessian_type(), Model::numFns, ActiveSet::request_vector(), Dakota::strbegins(), Model::surrogateType, and Interface::update_approximation().

Referenced by DataFitSurrModel::build_approximation().

13.19.3.18 void update_actual_model () [private]

update actualModel with data from current variables/labels/bounds/targets Update variables and constraints data within actualModel using values and labels from currentVariables and bound/linear/nonlinear constraints from userDefinedConstraints.

References Dakota::abort_handler(), DataFitSurrModel::actualModel, Model::aleatDistParams, Model::aleatory_distribution_parameters(), Model::all_continuous_lower_bounds(), Constraints::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Model::all_continuous_variable_labels(), Variables::all_continuous_variable_labels(), Model::all_continuous_variables(), Variables::all_continuous_variables(), Model::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_lower_bounds(), Model::all_discrete_int_variable_labels(), Variables::all_discrete_int_variable_labels(), Model::all_discrete_int_variables(), Variables::all_discrete_int_variables(), Model::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_lower_bounds(), Model::all_discrete_real_variable_labels(), Variables::all_discrete_real_variable_labels(), Model::all_discrete_real_variables(), Variables::all_discrete_real_variables(), SurrogateModel::approxBuilds, Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_variable_labels(), Variables::continuous_variables(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Model::cv(), Variables::cv(), Model::discrete_design_set_int_values(), Variables::discrete_design_set_real_values(), Constraints::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(),
update current variables/labels/bounds/targets with data from actualModel Update values and labels in current-
Variables and bound/linear/nonlinear constraints in userDefinedConstraints from variables and constraints data 
within actualModel.

References Dakota::abort_handler(), DataFitSurrModel::actualModel, Model::aleatDistParams, Model::aleatDistParams, 
Model::all_continuous_lower_bounds(), Constraints::all_continuous_lower_bounds(), Constraints::all_continuous_upper_ 
bounds(), Model::all_continuous_variable_labels(), Variables::all_continuous_variable_labels(), Model::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), 
Model::all_discrete_int_variable_labels(), Variables::all_discrete_int_variable_labels(), Model::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_upper_bounds(), 
Model::all_discrete_real_variable_labels(), Variables::all_discrete_real_variable_labels(), Model::all_discrete_real_variables(), SurrogateModel::approxBuilds, Model::currentResponse, Model::currentVariables, Variables::cv(), Variables::cv(), 
Model::discrete_design_set_int_values(), Model::discrete_design_set_real_values(), Model::discrete_state_set_int_values(), Model::discrete_state_set_real_values(), Model::discreteDesignSetIntValues, 
Model::discreteDesignSetRealValues, Model::discreteStateSetIntValues, Model::discreteStateSetRealValues, 
Model::div(), Variables::div(), Model::drv(), Variables::drv(), Model::epistDistParams, Model::epistemic_ 
distribution_parameters(), Response::function_labels(), Variables::inactive_continuous_variable_labels(), Variables::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(), 
Constraints::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_coeffs(), Constraints::linear_eq_constraint_coeffs(), 
Constraints::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_coeffs(), Model::linear_ineq_constraint_coeffs(), 
Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), 
Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), 
Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), 
Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), 
Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs() 
Model::response_labels(), Model::userDefinedConstraints, and Variables::view().

Referenced by DataFitSurrModel::build_approximation(), DataFitSurrModel::derived_async_compute_ response(), and DataFitSurrModel::derived_compute_response().

13.19.3.19 void update_from_actual_model () [private]
eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), Constraints::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), Constraints::linear_ineq_constraint_upper_bounds(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::num_linear_eq_constraints(), Model::num_linear_ineq_constraints(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primaryRespFnSense, Model::primaryRespFnWts, Model::response_labels(), and Model::userDefinedConstraints.

Referenced by DataFitSurrModel::DataFitSurrModel(), and DataFitSurrModel::update_from_subordinate_model().

13.19.4 Member Data Documentation

13.19.4.1 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::component_parallel_mode(), DataFitSurrModel::DataFitSurrModel(), DataFitSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_compute_response(), DataFitSurrModel::derived_free_communicators(), DataFitSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_serial(), DataFitSurrModel::derived_set_communicators(), DataFitSurrModel::derived_subordinate_models(), DataFitSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize_nowait(), DataFitSurrModel::fine_grained_evaluation_counters(), DataFitSurrModel::import_points(), DataFitSurrModel::inactive_view(), DataFitSurrModel::inside(), DataFitSurrModel::primary_response_fn_weights(), DataFitSurrModel::print_evaluation_summary(), DataFitSurrModel::serve(), DataFitSurrModel::stop_servers(), DataFitSurrModel::surrogate_response_mode(), DataFitSurrModel::truth_model(), DataFitSurrModel::update_actual_model(), DataFitSurrModel::update_from_actual_model(), DataFitSurrModel::update_from_subordinate_model(), DataFitSurrModel::update_global(), and DataFitSurrModel::update_local_multipoint().

The documentation for this class was generated from the following files:

- DataFitSurrModel.hpp
- DataFitSurrModel.cpp
13.20  **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*
- **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*

**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**
13.20 DataInterface Class Reference

- void run_dakota_data()
  
  `library_mode default data initializer`

### 13.20.1 Detailed Description

Handle class for interface specification data. The DataInterface class is used to provide a memory management handle for the data in DataInterfaceRep. 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
13.21 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

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
13.21 DataMethod Class Reference

- void run_dakota_data ()
  
  library_mode default data initializer

13.21.1 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::method_kwhandler() 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
13.22 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)

- **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 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
• RealVector linearIneqConstraintCoeffs
  coefficient matrix for the linear inequality constraints (from the linear_inequality_constraint_matrix specification in MethodIndControl)

• RealVector linearIneqLowerBnds
  lower bounds for the linear inequality constraints (from the linear_inequality_lower_bounds specification in MethodIndControl)

• RealVector linearIneqUpperBnds
  upper bounds for the linear inequality constraints (from the linear_inequality_upper_bounds specification in MethodIndControl)

• StringArray linearIneqScaleTypes
  scaling types for the linear inequality constraints (from the linear_inequality_scale_types specification in MethodIndControl)

• RealVector linearIneqScales
  scaling factors for the linear inequality constraints (from the linear_inequality_scales specification in MethodIndControl)

• RealVector linearEqConstraintCoeffs
  coefficient matrix for the linear equality constraints (from the linear_equality_constraint_matrix specification in MethodIndControl)

• RealVector linearEqTargets
  targets for the linear equality constraints (from the linear_equality_targets specification in MethodIndControl)

• StringArray linearEqScaleTypes
  scaling types for the linear equality constraints (from the linear_equality_scale_types specification in MethodIndControl)

• RealVector linearEqScales
  scaling factors for the linear equality constraints (from the linear_equality_scales specification in MethodIndControl)

• String methodName
  the method selection: one of the optimizer, least squares, nond, dace, or parameter study methods

• String subMethodName
  string identifier for a sub-method within a multi-option method specification (e.g., from sub_method_name in SBL/SBG, dace option, or richardson_extrap option)

• String subMethodPointer
  string pointer for a sub-method specification used by a multi-component method (from the sub_method_pointer specification in SBL/SBG)

• int surrBasedLocalSoftConvLimit
number of consecutive iterations with change less than convergenceTolerance required to trigger convergence within the surrogate-based local method (from the soft_convergence_limit specification in MethodSBL)

- bool surrBasedLocalLayerBypass
  flag to indicate user-specification of a bypass of any/all layerings in evaluating truth response values in SBL.

- Real surrBasedLocalTRInitSize
  initial trust region size in the surrogate-based local method (from the initial_size specification in MethodSBL) note: this is a relative value, e.g., 0.1 = 10% of global bounds distance (upper bound - lower bound) for each variable

- Real surrBasedLocalTRMinSize
  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 surrBasedLocalTRContractTrigger
  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 surrBasedLocalTRExpendTrigger
  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 surrBasedLocalTRContract
  trust region contraction factor in the surrogate-based local method (from the contraction_factor specification in MethodSBL)

- Real surrBasedLocalTRExpend
  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 strategy.

- 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 `constraintPenalty` 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., below_limit 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 max_designs 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 population_size 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 MethodFSU-DACE)

• IntVector sequenceStart
  the sequenceStart specification in MethodFSUDACE

• IntVector sequenceLeap
  the sequenceLeap specification in MethodFSUDACE
13.22 DataMethodRep Class Reference

- **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*

- **String historyFile**
  
  *the HISTORY_FILE specification for NOMAD*

- **String displayFormat**
  
  *the DISPLAY_STATS specification for NOMAD*

- **Real vns**
  
  *the VNS specification for NOMAD*

- **bool showAllEval**
  
  *the DISPLAY_ALL_EVAL 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 a strategy 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 a strategy with repeated sampling.*

- **int previousSamples**
  
  *the number of previous samples when augmenting a LHS sample*

- **bool vbdFlag**
  
  *the var_based_decomp specification for a variety of sampling methods*

- **Real vbdDropTolerance**
  
  *the var_based_decomp tolerance for omitting index output*

- **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 `sparseGridBasisType`  
  enumeration for type of basis in sparse grid interpolation: DEFAULT_INTERPOLANT, NODAL_INTERPOLANT, or HIERARCHICAL_INTERPOLANT

- UShortArray `expansionOrder`  
  the expansion_order specification in MethodNonDPCE

- SizetArray `expansionSamples`  
  the expansion_samples specification in MethodNonDPCE

- String `expansionSampleType`  
  allows for incremental PCE construction using the incremental_lhs specification in MethodNonDPCE

- UShortArray `quadratureOrder`  
  the quadrature_order specification in MethodNonDPCE and MethodNonDSC

- UShortArray `sparseGridLevel`  
  the sparse_grid_level specification in MethodNonDPCE, MethodNonDSC, and other stochastic expansion-enabled methods
- **RealVector** `anisoDimPref`  
  *the dimension_preference specification for tensor and sparse grids and expansion orders in MethodNonD-
  PCE and MethodNonDSC*

- **unsigned short** `cubIntOrder`  
  *the cubature_integrand specification in MethodNonDPCE*

- **SizetArray** `collocationPoints`  
  *the collocation_points 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** `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

• String expansionImportFile
  the expansion_import_file specification in MethodNonDPCE

• String sampleType
  the sample_type specification in MethodNonDMC, MethodNonDPCE, and MethodNonDSC

• String 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

• String integrationRefine
  the import, adapt_import, or mm_adapt_import integration refinement selection in MethodNonDLocalRel, MethodNonDPCE, and MethodNonDSC

• String nondOptAlgorithm
  the algorithm selection sqp or nip used for computing the MPP in MethodNonDLocalRel or the interval in MethodNonDLocalIntervalEst

• 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 emulatorSamples
the number of samples to construct a GP emulator for Bayesian calibration methods (MethodNonDBayesCalib)

- short emulatorType
  the emulator specification in MethodNonDBayesCalib

- String rejectionType
  the rejection type specification in MethodNonDBayesCalib

- String metropolisType
  the metropolis type specification in MethodNonDBayesCalib

- RealVector proposalCovScale
  the proposal covariance scale factor in MethodNonDBayesCalib

- Real likelihoodScale
  the likelihood scale factor in MethodNonDBayesCalib

- String fitnessMetricType
  the fitness metric type specification in MethodNonDAdaptive

- String batchSelectionType
  the batch selection type specification in MethodNonDAdaptive

- int batchSize
  The size of the batch (e.g. number of supplemental points added) to be added to be added to the build points for an emulator at each iteration.

- bool calibrateSigmaFlag
  flag to indicate if the sigma terms should be calibrated in MethodNonDBayesCalib

- 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

- 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

- bool pstudyFileAnnotated
  whether the parameter study points file is annotated

- UShortArray varPartitions
  the partitions specification for PStudy method in MethodPSMPS

- Real refinementRate
  rate of mesh refinement in Richardson extrapolation

- String approxImportFile
  the file name for point import in surrogate-based methods

- bool approxImportAnnotated
  whether the point import file is annotated (default true)

- String approxExportFile
  the file name for point export in surrogate-based methods

- bool approxExportAnnotated
  whether the point export file is annotated (default true)

Private Member Functions

- DataMethodRep ()
  constructor

- ~DataMethodRep ()
  destructor
13.22 DataMethodRep Class Reference

• 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

13.22.1 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
13.23 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

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
13.23 DataModel Class Reference

13.23.1 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
13.24 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)

- **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 truthModelPointer**
  - pointer to the model specification for constructing the truth model used in building local, multipoint, and hierarchical approximations (from the actual_model_pointer specification in ModelSurrL and ModelSurrMP and the high_fidelity_model_pointer specification in ModelSurrH)

- **String lowFidelityModelPointer**
  - pointer to the low fidelity model specification used in hierarchical approximations (from the low_fidelity_model_pointer 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 approxImportFile
  the file name from the import_points_file specification in ModelSurrG

• bool approxImportAnnotated
  whether the point import file is annotated (default true)

• String approxExportFile
  the file name from the export_points_file specification in ModelSurrG

• bool approxExportAnnotated
  whether the point export file is annotated (default true)

• String approxExportModelFile
  the file name from the export_model_file specification in ModelSurrG

• 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 mlsPolyOrder
  polynomial order for moving least squares approximation

• 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

• 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 approxChallengeFile
  the file name from the challenge_points_file specification in ModelSurrG

• bool approxChallengeAnnotated
  whether the challenge data file is annotated (default true)

• 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)

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
13.24 DataModelRep Class Reference

13.24.1 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`
13.25 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

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
13.25 DataResponses Class Reference

- `void run_dakota_data ()`

  *library_mode default data initializer*

13.25.1 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
13.26 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)

- StringArray 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 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 numLeastSqTerms
  number of least squares terms (from the num_least_squares_terms specification in RespFnLS)

- size_t numResponseFunctions
  number of generic response functions (from the num_response_functions specification in RespFnGen)

- 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)

- **size_t numExperiments**
  number of distinct experiments in experimental data

- **IntVector numReplicates**
  number of replicates in experimental data (e.g. one experiment run many times at the same configuration gives replicates)

- **size_t numExpConfigVars**
  number of experimental configuration vars (state variables) in each row of data

- **size_t numExpStdDeviations**
  whether to read num_responses standard deviations from each row of data file

- **RealVector expConfigVars**
  list of num_experiments x num_config_vars configuration variable values

- **RealVector expObservations**
  list of num_calibration_terms observation data

- **RealVector expStdDeviations**
  list of num_calibration_terms observation data
list of 1 or num_calibration_terms observation standard deviations

- **String expDataFileName**
  name of experimental data file containing response data (with optional state variable and sigma data) to read

- **bool expDataFileAnnotated**
  whether the experimental data is in annotated format

- **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)
13.26 DataResponsesRep Class Reference

- 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 id_analytic_gradients specification in RespGradMixed)

- IntSet idNumericalHessians
mixed Hessian numerical identifiers (from the id_numerical_hessians specification in RespHessMixed)

- IntSet idQuasiHessians
mixed Hessian quasi identifiers (from the id_quasi_hessians specification in RespHessMixed)

- IntSet idAnalyticHessians
mixed Hessian analytic identifiers (from the id_analytic_hessians specification in RespHessMixed)

Private Member Functions

- DataResponsesRep()
  constructor

- ~DataResponsesRep()
  destructor

- void write (std::ostream &s) const
  write a DataResponsesRep object to an std::ostream

- void read (MPIUnpackBuffer &s)
  read a DataResponsesRep object from a packed MPI buffer

- void write (MPIPackBuffer &s) const
  write a DataResponsesRep object to a packed MPI buffer

Private Attributes

- int referenceCount
  number of handle objects sharing this dataResponsesRep
Friends

- class DataResponses
  
  *the handle class can access attributes of the body class directly*

13.26.1 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
13.27 DataStrategy Class Reference

Handle class for strategy specification data.

Public Member Functions

- **DataStrategy ()**
  
  *constructor*

- **DataStrategy (const DataStrategy &)**
  
  *copy constructor*

- **~DataStrategy ()**
  
  *destructor*

- **DataStrategy & operator= (const DataStrategy &)**
  
  *assignment operator*

- **void write (std::ostream &s) const**
  
  *write a DataStrategy object to an std::ostream*

- **void read (MPIUnpackBuffer &s)**
  
  *read a DataStrategy object from a packed MPI buffer*

- **void write (MPIPackBuffer &s) const**
  
  *write a DataStrategy object to a packed MPI buffer*

Private Attributes

- **DataStrategyRep * dataStratRep**
  
  *pointer to the body (handle-body idiom)*

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

13.27.1 Detailed Description

Handle class for strategy specification data. The DataStrategy class is used to provide a memory management handle for the data in DataStrategyRep. It is populated by IDRProblemDescDB::strategy_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A single DataStrategy object is maintained in ProblemDescDB::strategySpec.
The documentation for this class was generated from the following files:

- DataStrategy.hpp
- DataStrategy.cpp
13.28 DataStrategyRep Class Reference

Body class for strategy specification data.

Public Attributes

- String **strategyType**
  
  *the strategy selection: hybrid, multi_start, pareto_set, or single_method*

- bool **graphicsFlag**
  
  *flags use of graphics by the strategy (from the graphics specification in StratIndControl)*

- bool **tabularDataFlag**
  
  *flags tabular data collection by the strategy (from the tabular_graphics_data specification in StratIndControl)*

- String **tabularDataFile**
  
  *the filename used for tabular data collection by the strategy (from the tabular_graphics_file specification in StratIndControl)*

- 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*

- int **iteratorServers**
  
  *number of servers for concurrent iterator parallelism (from the iterator_servers specification in StratIndControl)*

- String **iteratorScheduling**
  
  *type of scheduling (self or static) used in concurrent iterator parallelism (from the iterator_self_scheduling and iterator_static_scheduling specifications in StratIndControl)*

- String **methodPointer**
  
  *method identifier for the strategy (from the opt_method_pointer specifications in StratParetoSet and method_pointer specifications in StratSingle and StratMultiStart)*

- StringArray **hybridMethodList**
  
  *array of methods for the sequential and collaborative hybrid optimization strategies (from the method_list specification in StratHybrid)*

- String **hybridType**
  
  *the type of hybrid optimization strategy: collaborative, embedded, sequential, or sequential_adaptive (from the collaborative, embedded, and sequential specifications in StratHybrid)*
• **String hybridGlobalMethodPointer**
  global method pointer for embedded hybrids (from the `global_method_pointer` specification in `StratHybrid`)

• **String hybridLocalMethodPointer**
  local method pointer for embedded hybrids (from the `local_method_pointer` specification in `StratHybrid`)

• **Real hybridLSProb**
  local search probability for embedded hybrids (from the `local_search_probability` specification in `StratHybrid`)

• **int concurrentRandomJobs**
  number of random jobs to perform in the concurrent strategy (from the `random_starts` and `random_weight_sets` specifications in `StratMultiStart` and `StratParetoSet`)

• **int concurrentSeed**
  seed for the selected random jobs within the concurrent strategy (from the `seed` specification in `StratMultiStart` and `StratParetoSet`)

• **RealVector concurrentParameterSets**
  user-specified (i.e., nonrandom) parameter sets to evaluate in the concurrent strategy (from the `starting_points` and `multi_objective_weight_sets` specifications in `StratMultiStart` and `StratParetoSet`)

**Private Member Functions**

• **DataStrategyRep ()**
  constructor

• ~**DataStrategyRep ()**
  destructor

• **void write (std::ostream &s) const**
  write a `DataStrategyRep` object to an std::ostream

• **void read (MPIUnpackBuffer &s)**
  read a `DataStrategyRep` object from a packed MPI buffer

• **void write (MPIPackBuffer &s) const**
  write a `DataStrategyRep` object to a packed MPI buffer

**Private Attributes**

• **int referenceCount**
  number of handle objects sharing this dataStrategyRep
13.28 DataStrategyRep Class Reference

Friends

- class DataStrategy
  
  *the handle class can access attributes of the body class directly*

13.28.1 Detailed Description

Body class for strategy specification data. The DataStrategyRep class is used to contain the data from the strategy keyword specification. Default values are managed in the DataStrategyRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::strategySpec is private.

The documentation for this class was generated from the following files:

- DataStrategy.hpp
- DataStrategy.cpp
13.29 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
- `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
• void run_dakota_data ()
  library_mode default data initializer

13.29.1 Detailed Description

Handle class for variables specification data. The DataVariables class is used to provide a memory management handle for the data in DataVariablesRep. 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
13.30 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

- 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 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 numHistogramPtUncVars`
  number of 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 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 \texttt{discrete\_design\_range initial\_point} specification in \texttt{VarDV})

- \texttt{IntVector discreteDesignRangeLowerBnds}
  lower bounds array for the discrete design variables defined by an integer range (from the \texttt{discrete\_design\_range lower\_bounds} specification in \texttt{VarDV})

- \texttt{IntVector discreteDesignRangeUpperBnds}
  upper bounds array for the discrete design variables defined by an integer range (from the \texttt{discrete\_design\_range upper\_bounds} specification in \texttt{VarDV})

- \texttt{IntVector discreteDesignSetIntVars}
  initial values for the discrete design variables defined by an integer set (from the \texttt{discrete\_design\_set\_integer initial\_point} specification in \texttt{VarDV})

- \texttt{RealVector discreteDesignSetRealVars}
  initial values for the discrete design variables defined by a real set (from the \texttt{discrete\_design\_set\_real initial\_point} specification in \texttt{VarDV})

- \texttt{IntSetArray discreteDesignSetInt}
  complete set of admissible values for each of the discrete design variables defined by an integer set (from the \texttt{discrete\_design\_set\_integer set\_values} specification in \texttt{VarDV})

- \texttt{RealSetArray discreteDesignSetReal}
  complete set of admissible values for each of the discrete design variables defined by a real set (from the \texttt{discrete\_design\_set\_real set\_values} specification in \texttt{VarDV})

- \texttt{StringArray continuousDesignLabels}
  labels array for the continuous design variables (from the \texttt{continuous\_design descriptors} specification in \texttt{VarDV})

- \texttt{StringArray discreteDesignRangeLabels}
  labels array for the discrete design variables defined by an integer range (from the \texttt{discrete\_design\_range descriptors} specification in \texttt{VarDV})

- \texttt{StringArray discreteDesignSetIntLabels}
  labels array for the discrete design variables defined by an integer set (from the \texttt{discrete\_design\_range descriptors} specification in \texttt{VarDV})

- \texttt{StringArray discreteDesignSetRealLabels}
  labels array for the discrete design variables defined by a real set (from the \texttt{discrete\_design\_range descriptors} specification in \texttt{VarDV})

- \texttt{RealVector normalUncMeans}
  means of the normal uncertain variables (from the \texttt{nuv\_means} specification in \texttt{VarAUV})

- \texttt{RealVector normalUncStdDevs}

Dakota Version 5.4 Developers Manual generated on November 7, 2013
standard deviations of the normal uncertain variables (from the nuv_std_deviations specification in VarAUV)

- RealVector normalUncLowerBnds
distribution lower bounds for the normal uncertain variables (from the nuv_lower_bounds specification in VarAUV)

- RealVector normalUncUpperBnds
distribution upper bounds for the normal uncertain variables (from the nuv_upper_bounds specification in VarAUV)

- RealVector lognormalUncLambdas
  lambdas (means of the corresponding normals) of the lognormal uncertain variables (from the lnuv_lambdas specification in VarAUV)

- RealVector lognormalUncZetas
  zetas (standard deviations of the corresponding normals) of the lognormal uncertain variables (from the lnuv_zetas specification in VarAUV)

- RealVector lognormalUncMeans
  means of the lognormal uncertain variables (from the lnuv_means specification in VarAUV)

- RealVector lognormalUncStdDevs
  standard deviations of the lognormal uncertain variables (from the lnuv_std_deviations specification in VarAUV)

- RealVector lognormalUncErrFacts
  error factors for the lognormal uncertain variables (from the lnuv_error_factors specification in VarAUV)

- RealVector lognormalUncLowerBnds
distribution lower bounds for the lognormal uncertain variables (from the lnuv_lower_bounds specification in VarAUV)

- RealVector lognormalUncUpperBnds
distribution upper bounds for the lognormal uncertain variables (from the lnuv_upper_bounds specification in VarAUV)

- RealVector uniformUncLowerBnds
distribution lower bounds for the uniform uncertain variables (from the uuv_lower_bounds specification in VarAUV)

- RealVector uniformUncUpperBnds
distribution upper bounds for the uniform uncertain variables (from the uuv_upper_bounds specification in VarAUV)

- RealVector loguniformUncLowerBnds
distribution lower bounds for the loguniform uncertain variables (from the luuv_lower_bounds specification in VarAUV)
- RealVector `loguniformUncUpperBnds` 
  distribution upper bounds for the loguniform uncertain variables (from the `luuv_upper_bounds` specification in `VarAUV`)

- RealVector `triangularUncModes` 
  modes of the triangular uncertain variables (from the `tuv_modes` specification in `VarAUV`)

- RealVector `triangularUncLowerBnds` 
  distribution lower bounds for the triangular uncertain variables (from the `tuv_lower_bounds` specification in `VarAUV`)

- RealVector `triangularUncUpperBnds` 
  distribution upper bounds for the triangular uncertain variables (from the `tuv_upper_bounds` specification in `VarAUV`)

- RealVector `exponentialUncBetas` 
  beta factors for the exponential uncertain variables (from the `euv_betas` specification in `VarAUV`)

- RealVector `betaUncAlphas` 
  alpha factors for the beta uncertain variables (from the `buv_means` specification in `VarAUV`)

- RealVector `betaUncBetas` 
  beta factors for the beta uncertain variables (from the `buv_std_deviations` specification in `VarAUV`)

- RealVector `betaUncLowerBnds` 
  distribution lower bounds for the beta uncertain variables (from the `buv_lower_bounds` specification in `VarAUV`)

- RealVector `betaUncUpperBnds` 
  distribution upper bounds for the beta uncertain variables (from the `buv_upper_bounds` specification in `VarAUV`)

- RealVector `gammaUncAlphas` 
  alpha factors for the gamma uncertain variables (from the `gauv_alphas` specification in `VarAUV`)

- RealVector `gammaUncBetas` 
  beta factors for the gamma uncertain variables (from the `gauv_betas` specification in `VarAUV`)

- RealVector `gumbelUncAlphas` 
  alpha factors for the gumbel uncertain variables (from the `guuv_alphas` specification in `VarAUV`)

- RealVector `gumbelUncBetas` 
  beta factors for the gumbel uncertain variables (from the `guuv_betas` specification in `VarAUV`)

- RealVector `frechetUncAlphas` 
  alpha factors for the frechet uncertain variables (from the `fuv_alphas` specification in `VarAUV`)
- RealVector `frechetUncBetas`
  beta factors for the frechet uncertain variables (from the `fuv_betas` specification in `VarAUV`)

- RealVector `weibullUncAlphas`
  alpha factors for the weibull uncertain variables (from the `wuv_alphas` specification in `VarAUV`)

- RealVector `weibullUncBetas`
  beta factors for the weibull uncertain variables (from the `wuv_betas` specification in `VarAUV`)

- RealVectorArray `histogramUncBinPairs`
  an array containing a vector of (x,c) pairs for each bin-based histogram uncertain variable (see continuous linear histogram in LHS manual; from the `histogram_bin_uncertain` specification in `VarAUV`). (x,y) ordinate specifications are converted to (x,c) counts within NIDR.

- RealVector `poissonUncLambdas`
  lambdas (rate parameter) for the poisson uncertain variables (from the `lambdas` specification in `VarAUV`)

- RealVector `binomialUncProbPerTrial`
  probabilities per each trial (p) for the binomial uncertain variables from the `prob_per_trial` specification in `VarAUV`)

- IntVector `binomialUncNumTrials`
  Number of trials (N) for the binomial uncertain variables from the `num_trials` specification in `VarAUV`).

- RealVector `negBinomialUncProbPerTrial`
  probabilities per each trial (p) for the negative binomial uncertain variables from the `prob_per_trial` specification in `VarAUV`)

- IntVector `negBinomialUncNumTrials`
  Number of trials (N) for the negative binomial uncertain variables from the `num_trials` specification in `VarAUV`).

- RealVector `geometricUncProbPerTrial`
  probabilities per each trial (p) for the geometric uncertain variables from the `prob_per_trial` specification in `VarAUV`)

- IntVector `hyperGeomUncTotalPop`
  Size of total populations (N) for the hypergeometric uncertain variables from the `total_population` specification in `VarAUV`).

- IntVector `hyperGeomUncSelectedPop`
  Size of selected populations for the hypergeometric uncertain variables from the `selected_population` specification in `VarAUV`).

- IntVector `hyperGeomUncNumDrawn`
  Number failed in the selected populations for the hypergeometric variables from the `num_drawn` specification in `VarAUV`).
• RealVectorArray histogramUncPointPairs
  an array containing a vector of (x,c) pairs for each point-based histogram uncertain variable (see discrete histo-
  gram in LHS manual; from the histogram_point_uncertain specification in VarAUV)

• RealSymMatrix uncertainCorrelations
  correlation matrix for all uncertain variables (from the uncertain_correlation_matrix specification in VarAUV). This matrix specifies rank correlations for sampling methods (i.e., LHS) and correlation coefficients (rho_ij = normalized covariance matrix) for analytic reliability methods.

• RealVectorArray continuousIntervalUncBasicProbs
  Probability values per interval cell per epistemic interval uncertain variable (from the continuous IntervalUncBasicProbs specification in VarEUV).

• RealVectorArray continuousIntervalUncLowerBounds
  lower bounds defining cells for each epistemic interval uncertain variable (from the continuous_interval_uncertain lower_bounds specification in VarEUV)

• RealVectorArray continuousIntervalUncUpperBounds
  upper bounds defining cells for each epistemic interval uncertain variable (from the continuous_interval_uncertain upper_bounds specification in VarEUV)

• RealVectorArray discreteIntervalUncBasicProbs
  Probability values per interval cell per epistemic interval uncertain variable (from the discrete_interval_uncertain interval_probs specification in VarEUV).

• IntVectorArray discreteIntervalUncLowerBounds
  lower bounds defining cells for each epistemic interval uncertain variable (from the discrete_interval_uncertain lower_bounds specification in VarEUV)

• IntVectorArray discreteIntervalUncUpperBounds
  upper bounds defining cells for each epistemic interval uncertain variable (from the discrete_interval_uncertain upper_bounds specification in VarEUV)

• IntRealMapArray 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 VarEUV)

• RealRealMapArray 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 VarEUV)

• RealVector continuousStateVars
  initial values for the continuous state variables array (from the continuous_state initial_point specification in VarSV)
• RealVector `continuousStateLowerBnds`
  lower bounds array for the continuous state variables (from the `continuous_state lower_bounds` specification in `VarSV`)

• RealVector `continuousStateUpperBnds`
  upper bounds array for the continuous state variables (from the `continuous_state upper_bounds` specification in `VarSV`)

• IntVector `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`)

• 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`)

• 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`)

• 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`)

• 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_range descriptors` specification in `VarSV`)

- **StringArray `discreteStateSetRealLabels`**
  labels array for the discrete state variables defined by a real set (from the `discrete_state_range` 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

- **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`, and `buv_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

- 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

- 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`
  discrete state integer set lower bounds inferred from set values

- IntVector `discreteStateSetIntUpperBnds`
  discrete state integer set upper bounds inferred from set values

- RealVector `discreteStateSetRealLowerBnds`
  discrete state real set lower bounds inferred from set values

- RealVector `discreteStateSetRealUpperBnds`
  discrete state real set upper bounds inferred from set values

### Private Member Functions

- **DataVariablesRep ()**
  default constructor

- **~DataVariablesRep ()**
  destructor

- 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
13.30.1 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
13.31  DDACEDesignCompExp Class Reference

Wrapper class for the DDACE design of experiments library. Inheritance diagram for DDACEDesignCompExp::<br>

```
Iterator
  Analyzer
  PStudyDACE
  DDACEDesignCompExp
```

Public Member Functions

- **DDACEDesignCompExp (Model &model)**
  
  primary constructor for building a standard DACE iterator

- **DDACEDesignCompExp (Model &model, int samples, int symbols, int seed, const String &sampling_method)**
  
  alternate constructor used for building approximations

- **~DDACEDesignCompExp ()**
  
  destructor

- **void pre_run ()**
  
  pre-run portion of run_iterator (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

- **void extract_trends ()**
  
  Redefines the run_iterator virtual function for the PStudy/DACE branch.

- **void post_input ()**
  
  read tabular data for post-run mode

- **void post_run (std::ostream &s)**
  
  post-run portion of run_iterator (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**
  
  get the current number of samples

- **void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)**
  
  reset sampling iterator to use at least min_samples
• const String & 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)  
  Returns one block of samples (ndim * num_samples).

Private Member Functions

• void compute_main_effects ()  
  builds a DDaceMainEffects::OneWayANOVA if mainEffectsFlag is set

• void resolve_samples_symbols ()  
  convenience function for resolving number of samples and number of symbols from input.

Static Private Member Functions

• static void copy_data (const std::vector<DDaceSamplePoint> &dspa, Real *ptr, const int ptr_len)  
  copy DDACE point to RealVector

Private Attributes

• String daceMethod  
  oas, lhs, oa_lhs, random, box_behnken, 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_responses()

• size_t numDACERuns
  counter for number of run() executions for this object

• bool varyPattern
  flag for continuing the random number sequence from a previous run() 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

### 13.31.1 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.

### 13.31.2 Constructor & Destructor Documentation

#### 13.31.2.1 DDACEDesignCompExp (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::mainEffectsFlag, Iterator::maxConcurrency, Iterator::numContinuousVars, and DDACEDesignCompExp::numSamples.

#### 13.31.2.2 DDACEDesignCompExp (Model & model, int samples, int symbols, int seed, const String & 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: Iterator::maxConcurrency, DDACEDesignCompExp::numSamples, and DDACEDesignCompExp::resolve_samples_symbols().

### 13.31.3 Member Function Documentation

#### 13.31.3.1 void pre_run () [virtual]

Pre-run portion of run_iterator (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: DDACEDesignCompExp::get_parameter_sets(), Iterator::iteratedModel, and PStudyDACE::varBasedDecompFlag.

#### 13.31.3.2 void post_run (std::ostream & s) [virtual]

Post-run portion of run_iterator (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: Analyzer::allResponses, Analyzer::allSamples, SensAnalysisGlobal::compute_correlations(), DDACEDesignCompExp::compute_main_effects(), DDACEDesignCompExp::mainEffectsFlag, PStudyDACE::pStudyDACEsensGlobal, Iterator::subIteratorFlag, and PStudyDACE::varBasedDecompFlag.

#### 13.31.3.3 int num_samples () const [inline, virtual]

Get the current number of samples 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 maxConcurrency. May be (is) overridden by derived classes.

Reimplemented from Iterator.

References DDACEDesignCompExp::numSamples.

#### 13.31.3.4 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, Iterator::numContinuousVars, DDACEDesignCompExp::numSamples, and DDACEDesignCompExp::numSymbols.
Referenced by DDACEDesignCompExp::DDACEDesignCompExp(), and DDACEDesignCompExp::get_parameter_sets().

13.31.3.5 void copy_data (const std::vector< DDaceSamplePoint > & dspa, Real * ptr, const int ptr_len)  
[static, private]

copy DDACE point to RealVector copy DDACE point array to RealVectorArray copy DDACE point array to Real*

References Dakota::abort_handler().

Referenced by DDACEDesignCompExp::get_parameter_sets().

The documentation for this class was generated from the following files:

- DDACEDesignCompExp.hpp
- DDACEDesignCompExp.cpp
13.32 DirectApplicInterface Class Reference

Derived application interface class which spawns simulation codes and testers using direct procedure calls. Inheritance diagram for DirectApplicInterface:

```
  Interface
  ApplicationInterface
    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_iterator_concurrency)
- **void set_communicators_checks** (int max_iterator_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

- 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

- void set_local_data (const Response &response)
  
  convenience function for local test simulators which sets per-evaluation response attributes

- 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

- 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 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 xCLabels`
  continuous variable labels

- `StringMultiArray xDILabels`
  discrete integer variable labels

- `StringMultiArray xDRLabels`
  discrete real variable labels

- `std::map<String, var_t> varTypeMap`
  map from variable label to enum

- `std::map<String, driver_t> driverTypeMap`
  map from driver name to enum

- `std::map<var_t, Real> 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::vector< var_t > varTypeDVV`
  
  var_t enumerations corresponding to DVV components

• `std::vector< var_t > xCMLabels`
  
  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

• 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/ 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)
13.32.1 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.

13.32.2 Member Function Documentation

13.32.2.1 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().

13.32.2.2 void init_communicators_checks (int max_iterator_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().

13.32.2.3 void set_communicators_checks (int max_iterator_concurrency) [inline, virtual]

Process run-time issues as hard errors.
Reimplemented from ApplicationInterface.
Reimplemented in ParallelDirectApplicInterface, and SerialDirectApplicInterface.
References Dakota::abort_handler(), ApplicationInterface::check_asynchronous(), and ApplicationInterface::check_multiprocessor_asynchronous().

13.32.2.4 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 MatlabInterface, ParallelDirectApplicInterface, SerialDirectApplicInterface, PythonInterface, ScilabInterface, and TestDriverInterface.
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
13.33 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 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 apply (const Variables &vars, Response &approx_response, bool quiet_flag=false)`
  *apply the correction computed in compute() to approx_response*

- `bool active () const`
  *indicates an active correction via non-empty correctionType*

- `short correction_type () const`
  *return correctionType*
• short correction_order () const
  return correctionOrder

• short data_order () const
  return dataOrder

• bool computed () const
  return correctionComputed

Protected Attributes

• IntSet surrogateFnIndices
  for mixed response sets, this array specifies the response function subset that is approximated

• 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

- 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

### 13.33.1 Detailed Description

Base class for discrepancy corrections. The `DiscrepancyCorrection` class provides common functions for computing and applying corrections to approximations.

### 13.33.2 Member Function Documentation

#### 13.33.2.1 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::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::surrModel, DiscrepancyCorrection::surrogateFnIndices, DiscrepancyCorrection::truthFnsCenter, and DiscrepancyCorrection::truthFnsPrevCenter.

Referenced by HierarchSurrModel::derived_asynch_compute_response(), HierarchSurrModel::derived_compute_response(), DataFitSurrModel::derived_compute_response(), HierarchSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize(), HierarchSurrModel::derived_synchronize_nowait(), DataFitSurrModel::derived_synchronize_nowait(), and SurrBasedLocalMinimizer::minimize_surrogates().

The documentation for this class was generated from the following files:

- DiscrepancyCorrection.hpp
- DiscrepancyCorrection.cpp
13.34  DOTOptimizer Class Reference

Wrapper class for the DOT optimization library. Inheritance diagram for DOTOptimizer::

```
DOTOptimizer
 |               
|                
| Minimizer
 |               
|                
| Optimizer
 |               
|                
| Iterator
```

Public Member Functions

- **DOTOptimizer (Model &model)**
  
  *standard constructor*

- **DOTOptimizer (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor*

- **~DOTOptimizer ()**
  
  *destructor*

- **void find_optimum ()**
  
  *Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- **void initialize_run ()**
  
  *performs run-time set up*

Private Member Functions

- **void initialize ()**
  
  *Shared constructor code.*

- **void allocate_workspace ()**
  
  *Allocates workspace for the optimizer.*
void allocate_constraints ()
    Allocates constraint mappings.

Private Attributes

• int dotInfo
    INFO from DOT manual.

• int dotFDSinfo
    internal DOT parameter NGOTOZ

• int dotMethod
    METHOD from DOT manual.

• int printControl
    IPRINT from DOT manual (controls output verbosity).

• RealArray realCntlParmArray
    RPRM from DOT manual.

• IntArray intCntlParmArray
    IPRM from DOT manual.

• RealVector designVars
    array of design variable values passed to DOT

• Real objFnValue
    value of the objective function passed to DOT

• RealVector constraintValues
    array of nonlinear constraint values passed to DOT

• int realWorkSpaceSize
    size of realWorkSpace

• int intWorkSpaceSize
    size of intWorkSpace

• RealArray realWorkSpace
    real work space for DOT

• IntArray intWorkSpace
    int work space for DOT

• int numDotNlnConstr
13.34 DOTOptimizer Class Reference

- **int numDotLinConstr**
  total number of linear constraints seen by DOT

- **int numDotConstr**
  total number of linear and nonlinear constraints seen by DOT

- **SizetArray constraintMappingIndices**
  a container of indices for referencing the corresponding Response constraints used in computing the DOT constraints.

- **RealArray constraintMappingMultipliers**
  a container of multipliers for mapping the Response constraints to the DOT constraints.

- **RealArray constraintMappingOffsets**
  a container of offsets for mapping the Response constraints to the DOT constraints.

### 13.34.1 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 find_optimum() 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.

### 13.34.2 Member Data Documentation

#### 13.34.2.1 int dotInfo [private]

INFO from DOT manual. Information requested by DOT: 0=optimization complete, 1=get values, 2=get gradients

Referenced by DOTOptimizer::find_optimum(), and DOTOptimizer::initialize_run().

#### 13.34.2.2 int dotFDSinfo [private]

internal DOT parameter NGOTOZ the DOT parameter list has been modified to pass NGOTOZ, which signals whether DOT is finite-differencing (nonzero value) or performing the line search (zero value).

Referenced by DOTOptimizer::find_optimum().
13.34.2.3 int dotMethod [private]

METHOD from DOT manual. For nonlinear constraints: 0/1 = dot_mmfd, 2 = dot_slp, 3 = dot_sqp. For unconstrained: 0/1 = dot_bfgs, 2 = dot_frcg.

Referenced by DOTOptimizer::allocate_constraints(), DOTOptimizer::allocate_workspace(), DOTOptimizer::DOTOptimizer(), and DOTOptimizer::find_optimum().

13.34.2.4 int printControl [private]

IPRINT from DOT manual (controls output verbosity). Values range from 0 (least output) to 7 (most output).

Referenced by DOTOptimizer::DOTOptimizer(), and DOTOptimizer::find_optimum().

13.34.2.5 RealArray realCntlParmArray [private]

RPRM from DOT manual. Array of real control parameters.

Referenced by DOTOptimizer::find_optimum(), and DOTOptimizer::initialize().

13.34.2.6 IntArray intCntlParmArray [private]

IPRM from DOT manual. Array of integer control parameters.

Referenced by DOTOptimizer::find_optimum(), and DOTOptimizer::initialize().

13.34.2.7 RealVector constraintValues [private]

array of nonlinear constraint values passed to DOT This array must be of nonzero length and must contain only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities and equalities).

Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().

13.34.2.8 SizetArray constraintMappingIndices [private]

a container of indices for referencing the corresponding Response constraints used in computing the DOT constraints. The length of the container corresponds to the number of DOT constraints, and each entry in the container points to the corresponding DAKOTA constraint.

Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().

13.34.2.9 RealArray constraintMappingMultipliers [private]

a container of multipliers for mapping the Response constraints to the DOT constraints. The length of the container corresponds to the number of DOT constraints, and each entry in the container stores a multiplier for the DAKOTA constraint identified with constraintMappingIndices. These multipliers are currently +1 or -1.

Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().
13.34.2.10 RealArray constraintMappingOffsets [private]

a container of offsets for mapping the Response constraints to the DOT constraints. The length of the container corresponds to the number of DOT constraints, and each entry in the container stores an offset for the DAKOTA constraint identified with constraintMappingIndices. These offsets involve inequality bounds or equality targets, since DOT assumes constraint allowables = 0.

Referenced by DOTOptimizer::allocate_constraints(), and DOTOptimizer::find_optimum().

The documentation for this class was generated from the following files:

- DOTOptimizer.hpp
- DOTOptimizer.cpp
### 13.35 Driver Class Reference

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

#### 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.

### 13.35.1 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 find_optimum.

### 13.35.2 Constructor & Destructor Documentation

#### 13.35.2.1 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.

### 13.35.3 Member Function Documentation

#### 13.35.3.1 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::find_optimum().

13.35.3.2 DesignOfSortSet PerformIterations (GeneticAlgorithm * theGA) [inline]

Performs the required iterations on the supplied GA. This includes the calls to AlgorithmInitialize and Algorithm-Finalize 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::find_optimum().

13.35.3.3 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::find_optimum().

The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp
13.36 EffGlobalMinimizer Class Reference

Implementation of Efficient Global Optimization/Least Squares algorithms. Inheritance diagram for EffGlobalMinimizer:

```
EffGlobalMinimizer
  |__________________________|
  |                           |
  |     SurrBasedMinimizer    |
  |                           |
  |     Minimizer             |
  |                           |
  |     Iterator              |
```

Public Member Functions

- **EffGlobalMinimizer (Model &model)**
  - Standard constructor

- **~EffGlobalMinimizer ()**
  - Alternate constructor for instantiations "on the fly"

- **void minimize_surrogates ()**
  - Used for computing the optimal solution using a surrogate-based approach. Redefines the Iterator::run() virtual function.

- **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 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 expectedViolation (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
13.36.1 Detailed Description

Implementation of Efficient Global Optimization/Least Squares algorithms. 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.

13.36.2 Constructor & Destructor Documentation

13.36.2.1 ~EffGlobalMinimizer()

alternate constructor for instantiations "on the fly" destructor
This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.
References SurrBasedMinimizer::approxSubProbMinimizer, EffGlobalMinimizer::eifModel, Model::free_ communicators(), and Iterator::maximum_concurrency().

13.36.3 Member Function Documentation

13.36.3.1 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::compute_response(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), EffGlobalMinimizer::fHatModel, Response::function_values(), Iterator::iteratedModel, EffGlobalMinimizer::meritFnStar, Iterator::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
13.37 EfficientSubspaceMethod Class Reference

Efficient Subspace Method (ESM), as proposed by Hany S. Abdel-Khalik. Inheritance diagram for EfficientSubspaceMethod::

```
EfficientSubspaceMethod
  ^
  |  
  |  Analyzer
  |    
  |    NonD
  |      
  |      EfficientSubspaceMethod
  +----------------+
    Iterator
```

Public Member Functions

- **EfficientSubspaceMethod (Model &model)**
  
  *Standard, model-based constructor.*

- **~EfficientSubspaceMethod ()**
  
  *Destructor.*

- **void quantify_uncertainty ()**
  
  *ESM re-implementation of the virtual UQ iterator function.*

Private Member Functions

- **void validate_inputs ()**
  
  *validate user-supplied input values, setting defaults, aborting on error*

- **void init_fullspace_sampler ()**
  
  *initialize the native problem space Monte Carlo sampler*

- **void expand_basis (bool &mach_svtol_met, bool &user_svtol_met)**
  
  *generate fullspace samples, append to matrix, and factor, returning whether tolerance met*

- **unsigned int calculate_fullspace_samples ()**
  
  *determine the number of full space samples for next iteration, based on batchSize, limiting by remaining function evaluation budget*

- **void generate_fullspace_samples (unsigned int diff_samples)**
  
  *sample the derivative at diff_samples points and leave temporary in dace_iterator*
void append_sample_matrices (unsigned int diff_samples)
append the fullSpaceSampler samples to the derivative and vars matrices

void compute_svd (bool &mach_svtol_met, bool &user_svtol_met)
factor the derivative matrix and analyze singular values, assessing convergence and rank, returning whether tolerance met

void print_svd_stats ()
print inner iteration stats after SVD

void assess_reconstruction (bool &recon_tol_met)
determine if the reduced basis yields acceptable reconstruction error, based on sampling in the orthogonal complement of the reduced basis

void reduced_space_uq ()
experimental method to demonstrate creating a RecastModel and perform sampling-based UQ in the reduced space

void uncertain_vars_to_subspace (Model &native_model, Model &vars_transform_model)
translate the characterization of uncertain variables in the native_model to the reduced space of the transformed model

Static Private Member Functions

static void map_xi_to_x (const Variables &recast_xi_vars, Variables &sub_model_x_vars)
map the active continuous recast variables to the active submodel variables (linear transformation)

Private Attributes

int initialSamples
initial number of samples at which to query the truth model

int batchSize
number of points to add at each iteration

int subspaceSamples
number of UQ samples to perform in the reduced space

unsigned int currIter
current iteration

unsigned int totalSamples
total construction samples evaluated so far
• unsigned int totalEvals
  
  total evaluations of model (accounting for UQ phase)

• double userSVTol
  
  user-specified tolerance on singular value ratio

• double nullspaceTol
  
  user-specified tolerance on nullspace

• double svRatio
  
  current singular value ratio (sigma_k/sigma_0)

• unsigned int reducedRank
  
  current approximation of system rank

• RealMatrix reducedBasis
  
  basis for the reduced subspace

• 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) \times [D1
  \mid D2 \mid \ldots \mid D num\_ samples] \times [dy_1/\partial x(k=1) \mid dy_2/\partial x(k=1) \mid \ldots \mid dy_M/\partial x(k=1) \mid k=2 \mid \ldots \mid k=n_s]

• RealMatrix varsMatrix
  
  matrix of fullspace variable points samples size numContinuousVars \times (numSamples)

• Iterator fullSpaceSampler
  
  Monte Carlo sampler for the full parameter space.

Static Private Attributes

• static EfficientSubspaceMethod * esmInstance
  
  instance of this class for use in static member functions

13.37.1 Detailed Description

Efficient Subspace Method (ESM), as proposed by Hany S. Abdel-Khalik. ESM uses random sampling to con-
stuct a low-dimensional subspace of the full dimensional parameter space, then performs UQ in the reduced space
13.37.2 Member Function Documentation

13.37.2.1 void assess_reconstruction (bool & recon_tol_met) [private]

determine if the reduced basis yields acceptable reconstruction error, based on sampling in the orthogonal complement of the reduced basis. This function is experimental and needs to be carefully reviewed and cleaned up.

References Iterator::activeSet, Model::aleatory_distribution_parameters(), Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_values(), Iterator::iteratedModel, Iterator::maxFunctionEvals, EfficientSubspaceMethod::nullspaceTol, Iterator::numContinuousVars, Iterator::numFunctions, Iterator::outputLevel, EfficientSubspaceMethod::nullspaceTol, ActiveSet::request_values(), EfficientSubspaceMethod::totalEvals, EfficientSubspaceMethod::totalSamples, and EfficientSubspaceMethod::varsMatrix.

Referenced by EfficientSubspaceMethod::quantify_uncertainty().

13.37.2.2 void reduced_space_uq () [private]

experimental method to demonstrate creating a RecastModel and perform sampling-based UQ in the reduced space. This function is experimental and needs to be reviewed and cleaned up. In particular, the translation of the correlations from full to reduced space is likely wrong. Transformation may be correct for covariance, but likely not correlations.

References Model::assign_rep(), NonD::construct_lhs(), Model::init_communicators(), Iterator::iteratedModel, EfficientSubspaceMethod::map_xi_to_x(), Iterator::numContinuousVars, Iterator::numFunctions, Iterator::print_results(), EfficientSubspaceMethod::reducedRank, Iterator::run_iterator(), Iterator::sampling_reset(), Iterator::sub_iterator_flag(), EfficientSubspaceMethod::subspaceSamples, EfficientSubspaceMethod::uncertain_vars_to_subspace(), and Analyzer::vary_pattern().

Referenced by EfficientSubspaceMethod::quantify_uncertainty().

13.37.2.3 void uncertain_vars_to_subspace (Model & native_model, Model & vars_transform_model) [private]

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

References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Iterator::numContinuousVars, Iterator::outputLevel, EfficientSubspaceMethod::reducedBasis, and EfficientSubspaceMethod::reducedRank.

Referenced by EfficientSubspaceMethod::reduced_space_uq().

13.37.2.4 void map_xi_to_x (const Variables & recast_xi_vars, Variables & sub_model_x_vars) [static, private]

map the active continuous recast variables to the active submodel variables (linear transformation). Perform the variables mapping from recast reduced dimension variables $xi$ to original model $x$ variables via linear transforma-
tion. Maps only continuous variables.

References Variables::continuous_variables(), Dakota::copy_data(), EfficientSubspaceMethod::esmInstance, Iterator::outputLevel, and EfficientSubspaceMethod::reducedBasis.

Referenced by EfficientSubspaceMethod::reduced_space_uq().

The documentation for this class was generated from the following files:

- EfficientSubspaceMethod.hpp
- EfficientSubspaceMethod.cpp
13.38 EmbeddedHybridStrategy Class Reference

Strategy for closely-coupled hybrid minimization, typically involving the embedding of local search methods within global search methods. Inheritance diagram for EmbeddedHybridStrategy:

```
Strategy
   |        |
   |        |
HybridStrategy
   |        |
   |        |
EmbeddedHybridStrategy
```

Public Member Functions

- **EmbeddedHybridStrategy** (ProblemDescDB &problem_db)
  *constructor*

- **~EmbeddedHybridStrategy** ()
  *destructor*

Protected Member Functions

- void **run_strategy** ()
  
  *Performs the hybrid minimization strategy by executing multiple iterators on different models of varying fidelity.*

- const **Variables** & **variables_results** () const
  
  *return the final solution from selectedIterators (variables)*

- const **Response** & **response_results** () const
  
  *return the final solution from selectedIterators (response)*

Private Attributes

- Real **localSearchProb**
  
  *the probability of running a local search refinement within phases of the global minimization for coupled hybrids*

13.38.1 Detailed Description

Strategy for closely-coupled hybrid minimization, typically involving the embedding of local search methods within global search methods. This strategy 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:

- EmbeddedHybridStrategy.hpp
- EmbeddedHybridStrategy.cpp
13.39 Evaluator Class Reference

NOMAD-based Evaluator class.

Public Member Functions

- **Evaluator** (const NOMAD::Parameters &p, Model &model)
  Constructor.

- ~Evaluator (void)
  Destructor.

- bool eval_x (NOMAD::Eval_Point &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

Private Attributes

- Model & _model
- int n_cont
- int n_disc_int
- int n_disc_real
- int numNomadNonlinearIneqConstr
- int numNomadNonlinearEqConstr
- std::vector<int> constrMapIndices
  map from Dakota constraint number to APPS constraint number

- std::vector<double> constrMapMultipliers
  multipliers for constraint transformations

- std::vector<double> constrMapOffsets
  offsets for constraint transformations

13.39.1 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 NomadEvaluator 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.

### 13.39.2 Constructor & Destructor Documentation

#### 13.39.2.1 Evaluator (const NOMAD::Parameters & p, Model & model)

Constructor. NOMAD Evaluator Constructor

**Parameters:**
- `p` NOMAD Parameters object
- `model` DAKOTA Model object

### 13.39.3 Member Function Documentation

#### 13.39.3.1 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:**
- `x` Object that contains the points that need to evaluated. Once the evaluation is completed, this object also stores the output back to be read by NOMAD.
- `h_max` Current value of the barrier parameter. Not used in this implementation.
- `count_eval` Flag that indicates whether this evaluation counts towards the max number of evaluations, often set to false when the evaluation does not meet certain costs during expensive evaluations. Not used in this implementation.

**Returns:**
- `true` if the evaluation was successful; `false` otherwise.

References Model::compute_response(), Evaluator::constrMapIndices, Evaluator::constrMapMultipliers, Evaluator::constrMapOffsets, Model::continuous_variable(), Model::current_response(), Model::cv(), Model::discrete_int_sets(), Model::discrete_int_variable(), Model::discrete_real_variable(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::div(), Model::drv(), Response::function_values(), Model::primary_response_fn_sense(), and Dakota::set_index_to_value().

The documentation for this class was generated from the following files:

- NomadOptimizer.hpp
- NomadOptimizer.cpp
13.40  Evaluator Class Reference

An evaluator specialization that knows how to interact with Dakota.

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.

Protected Member Functions

- void SeparateVariables (const Design &from, RealVector &intoCont, IntVector &intoDiscInt, RealVector &intoDiscReal) 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.

13.40.1 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.

13.40.2 Constructor & Destructor Documentation

13.40.2.1 Evaluator (GeneticAlgorithm & algorithm, Model & model) [inline]

Constructs a Evaluator for use by algorithm. The optimizer is needed for purposes of variable scaling.

Parameters:

  algorithm The GA for which the new evaluator is to be used.

  model The model through which evaluations will be done.

Referenced by Evaluator::Clone().
13.40.2.2 Evaluator (const Evaluator & copy)  [inline]

Copy constructs a Evaluator.

Parameters:

  
  copy The evaluator from which properties are to be duplicated into this.

13.40.2.3 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:

  
  copy The existing Evaluator from which to retrieve properties.
  
  algorithm The GA for which the new evaluator is to be used.
  
  model The model through which evaluations will be done.

13.40.2.4 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:

  
  algorithm The GA for which the new evaluator is to be used.

13.40.3 Member Function Documentation

13.40.3.1 static const std::string& Name ()  [inline, static]

Returns the proper name of this operator.

Returns:

  
  The string "DAKOTA JEGA Evaluator".

Referenced by Evaluator::GetName().

13.40.3.2 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.

Referenced by Evaluator::GetDescription().

13.40.3.3 void SeparateVariables (const Design & from, RealVector & intoCont, IntVector & intoDiscInt, RealVector & intoDiscReal) 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:

from  The Design class object from which to extract the discrete design variable values.
intoDiscInt  The vector into which to place the extracted discrete integer values.
intoDiscReal  The vector into which to place the extracted discrete real values.
intoCont  The vector into which to place the extracted continuous values.

References Evaluator::_model, Model::cv(), Model::discrete_int_sets(), Model::div(), and Model::drv().
Referenced by Evaluator::Evaluate().

13.40.3.4 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:

from  The vector of responses to install into into.
into  The Design to which the responses belong and into which they must be written.

References Evaluator::GetNumberNonLinearConstraints().
Referenced by Evaluator::Evaluate().

13.40.3.5 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.

References Evaluator::_model, Model::num_nonlinear_eq_constraints(), and Model::num_nonlinear_ineq_constraints().

Referenced by Evaluator::Evaluate(), and Evaluator::RecordResponses().

13.40.3.6 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.

References Evaluator::_model, Model::num_linear_eq_constraints(), and Model::num_linear_ineq_constraints().

13.40.3.7 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.

References Evaluator::_model, Model::asynch_compute_response(), Model::asynch_flag(), Model::compute_response(), Model::continuous_variables(), Model::current_response(), Model::discrete_int_variables(), Model::discrete_real_variables(), Response::function_values(), Evaluator::GetName(), Evaluator::GetNumberNonLinearConstraints(), Evaluator::RecordResponses(), Evaluator::SeparateVariables(), and Model::synchronize().

13.40.3.8 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:

\textit{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.

References Evaluator::GetName().

13.40.3.9 \textbf{virtual std::string GetName () const} \texttt{[inline, virtual]}

Returns the proper name of this operator.

Returns:

See Name().

References Evaluator::Name().
Referenced by Evaluator::Evaluate().

13.40.3.10 \textbf{virtual std::string GetDescription () const} \texttt{[inline, virtual]}

Returns a full description of what this operator does and how.

Returns:

See Description().

References Evaluator::Description().

13.40.3.11 \textbf{virtual GeneticAlgorithmOperator* Clone (GeneticAlgorithm & \textit{algorithm}) const} \texttt{[inline, virtual]}

Creates and returns a pointer to an exact duplicate of this operator.

Parameters:

\textit{algorithm}  The GA for which the clone is being created.

Returns:

A clone of this operator.

References Evaluator::_model, and Evaluator::Evaluator().
13.40.4 Member Data Documentation

13.40.4.1 Model& _model [private]

The Model known by this evaluator. It is through this model that evaluations will take place.

Referenced by Evaluator::Clone(), Evaluator::Evaluate(), Evaluator::GetNumberLinearConstraints(), Evaluator::GetNumberNonLinearConstraints(), and Evaluator::SeparateVariables().

The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp
13.41 EvaluatorCreator Class Reference

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

Public Member Functions

- virtual GeneticAlgorithmEvaluator ∗ CreateEvaluator (GeneticAlgorithm &alg)
  
  Overridden to return a newly created Evaluator.

- EvaluatorCreator (Model &theModel)
  
  Constructs an EvaluatorCreator using the supplied model.

Private Attributes

- Model & _theModel
  
  The user defined model to be passed to the constructor of the Evaluator.

13.41.1 Detailed Description

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

13.41.2 Constructor & Destructor Documentation

13.41.2.1 EvaluatorCreator (Model & theModel) [inline]

Constructs an EvaluatorCreator using the supplied model.

Parameters:

  theModel  The Dakota::Model this creator will pass to the created evaluator.

13.41.3 Member Function Documentation

13.41.3.1 virtual GeneticAlgorithmEvaluator ∗ CreateEvaluator (GeneticAlgorithm & alg) [inline, virtual]

Overridden 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.

References EvaluatorCreator::_theModel.

The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp
13.42 ExperimentData Class Reference

Public Member Functions

- void load_scalar (const std::string &expDataFilename, const std::string &context_message, size_t numExperiments, IntVector &numReplicates, size_t numExpConfigVars, size_t numFunctions, size_t numExpStdDeviationsRead, bool expDataFileAnnotated, bool calc_sigma_from_data, short verbosity)
  
  constructor

- const RealVector & config_vars (size_t response, size_t experiment)
- Real scalar_data (size_t response, size_t experiment, size_t replicate)
- Real scalar_sigma (size_t response, size_t experiment, size_t replicate)

Public Attributes

- std::vector< ExpDataPerResponse > allExperiments

  At the outer level, ExperimentData will just be a vector of ExpDataPerResponse.

13.42.1 Detailed Description

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, number of replicates, 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.

13.42.2 Member Function Documentation

13.42.2.1 void load_scalar (const std::string &expDataFilename, const std::string &context_message, size_t numExperiments, IntVector &numReplicates, size_t numExpConfigVars, size_t numFunctions, size_t numExpStdDeviationsRead, bool expDataFileAnnotated, bool calc_sigma_from_data, short verbosity)

creator Constructor from legacy file format

References ExperimentData::allExperiments, and Dakota::read_historical_data().

Referenced by Minimizer::data_transform_model(), NonDQUESOBayesCalibration::quantify_uncertainty(), NonDGPMSABayesCalibration::quantify_uncertainty(), and NonDDREAMBayesCalibration::quantify_-_uncertainty().

The documentation for this class was generated from the following files:

- ExperimentData.hpp
- ExperimentData.cpp
13.43 ForkApplicInterface Class Reference

Derived application interface class which spawns simulation codes using fork/execvp/waitpid. Inheritance diagram for ForkApplicInterface::

```
Interface

ApplicationInterface

ProcessApplicInterface

ProcessHandleApplicInterface

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()/vfork()/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 asynch evaluations

- void join_analysis_process_group (bool new_group)
  create (if new_group) and join the process group for asynch analyses

- void evaluation_process_group_id (pid_t pgid)
  set evalProcGroupId

- pid_t evaluation_process_group_id () const
  return evalProcGroupId

- 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 evalProcGroupId
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 analysisProcGroupId`
  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)

### 13.43.1 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`
Wrapper class for the FSUDace QMC/CVT library. Inheritance diagram for FSUDesignCompExp::

Public Member Functions

- `FSUDesignCompExp (Model &model)`
  primary constructor for building a standard DACE iterator

- `FSUDesignCompExp (Model &model, int samples, int seed, const String &sampling_method)`
  alternate constructor for building a DACE iterator on-the-fly

- `~FSUDesignCompExp ()`
  destructor

- `void pre_run ()`
  pre-run portion of run_iterator (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

- `void extract_trends ()`
  Redefines the run_iterator virtual function for the PStudy/DACE branch.

- `void post_input ()`
  read tabular data for post-run mode

- `void post_run (std::ostream &s)`
  post-run portion of run_iterator (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`
  get the current number of samples

- `void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)`
  reset sampling iterator to use at least min_samples
• const String & 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)
  Returns one block of samples (ndim * num_samples).

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 run() 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.).
13.44 FSUDesignCompExp Class Reference

- **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 ran() 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.

### 13.44.1 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.

### 13.44.2 Constructor & Destructor Documentation

#### 13.44.2.1 FSUDesignCompExp (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::maxConcurrency, Iterator::methodName, Iterator::numContinuousVars, FSUDesignCompExp::numCVTTrials, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, Iterator::probDescDB, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, FSUDesignCompExp::trialType, and FSUDesignCompExp::varyPattern.

#### 13.44.2.2 FSUDesignCompExp (Model & model, int samples, int seed, const String & 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::maxConcurrency, Iterator::methodName, Iterator::numContinuousVars, FSUDesignCompExp::numCVTTrials, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, and FSUDesignCompExp::trialType.

13.44.3 Member Function Documentation

13.44.3.1 void pre_run () [virtual]

pre-run portion of run_iterator (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 FSUDesignCompExp::get_parameter_sets(), Iterator::iteratedModel, and PStudyDACE::varBasedDecompFlag.

13.44.3.2 void post_run (std::ostream & s) [virtual]

post-run portion of run_iterator (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 Analyzer::allResponses, Analyzer::allSamples, SensAnalysisGlobal::compute_correlations(), PStudyDACE::pStudyDACESensGlobal, Iterator::subIteratorFlag, and PStudyDACE::varBasedDecompFlag.

13.44.3.3 int num_samples () const [inline, virtual]

get the current number of samples 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 maxConcurrency. May be (is) overridden by derived classes.

Reimplemented from Iterator.

References FSUDesignCompExp::numSamples.

13.44.3.4 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, Iterator::numContinuousVars, FSUDesignCompExp::numSamples, and FSUDesignCompExp::primeBase.
Referenced by FSUDesignCompExp::get_parameter_sets().

The documentation for this class was generated from the following files:

- FSUDesignCompExp.hpp
- FSUDesignCompExp.cpp
13.45 GaussProcApproximation Class Reference

Derived approximation class for Gaussian Process implementation. Inheritance diagram for GaussProcApproximation:

```
Approximation
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GaussProcApproximation</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- `GaussProcApproximation ()`
  - default constructor

- `GaussProcApproximation (size_t num_vars, short data_order, short output_level)`
  - alternate constructor

- `GaussProcApproximation (const ProblemDescDB &problem_db, size_t num_vars)`
  - 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 ()`
  - 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()
  
  Calculate 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 &)
  
  Return the maximum value of the elements in a vector.

• void pointsel_write_points()
  
  Writes out the training set before and after point selection.

• 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

Static Private Attributes

• static GaussProcApproximation * GPinstance
  
  pointer to the active object instance used within the static evaluator

13.45.1 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.

13.45.2 Constructor & Destructor Documentation

13.45.2.1 GaussProcApproximation () [inline]

default constructor alternate constructor used by EffGlobalOptimization and NonDGlobalReliability that does not use a problem database defaults here are no point selection and quadratic trend function.

13.45.3 Member Function Documentation

13.45.3.1 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(), Approximation::numVars, GaussProcApproximation::predict(), GaussProcApproximation::trainMeans, and GaussProcApproximation::trainStdvs.

Referenced by GaussProcApproximation::gradient(), GaussProcApproximation::pointsel_get_errors(), GaussProcApproximation::prediction_variance(), and GaussProcApproximation::value().

13.45.4 Member Data Documentation

13.45.4.1 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
13.46 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 & 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

13.46.1 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 "=".

13.46.2 Member Enumeration Documentation

13.46.2.1 enum OptType

enum for different types of values associated with command line options.

**Enumerator:**

- **Valueless** option that may never have a value
13.46 GetLongOpt Class Reference

OptionalValue  option with optional value
MandatoryValue  option with required value

13.46.3  Constructor & Destructor Documentation

13.46.3.1  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.
References GetLongOpt::enroll_done, GetLongOpt::last, GetLongOpt::optmarker, GetLongOpt::table, and Get-
LongOpt::ustring.

13.46.4  Member Function Documentation

13.46.4.1  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.
References GetLongOpt::basename(), GetLongOpt::enroll_done, GetLongOpt::optmarker, GetLongOpt::pname,
GetLongOpt::setcell(), and GetLongOpt::table.
Referenced by CommandLineHandler::check_usage().

13.46.4.2  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.

13.46.4.3  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 sting. 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().

Dakota Version 5.4 Developers Manual generated on November 7, 2013
13.46.4.4  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::assign_streams(), CommandLineHandler::check_usage(), CommandLineHandler::instantiate_flag(), main(), ProblemDescDB::manage_inputs(), ParallelLibrary::manage_run_modes(), CommandLineHandler::read_restart_evals(), CommandLineHandler::reset_streams(), CommandLineHandler::run_flag(), and ParallelLibrary::specify_outputs_restart().

13.46.4.5  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
13.47 Graphics Class Reference

The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics as well as tabular cataloguing of data for post-processing with Matlab, Tecplot, etc.

Public Member Functions

- **Graphics ()**
  
  constructor

- **~Graphics ()**
  
  destructor

- **void create_plots_2d (const Variables &vars, const Response &response)**

  creates the 2d graphics window and initializes the plots

- **void create_tabular_datastream (const Variables &vars, const Response &response, const std::string &tabular_data_file)**

  opens the tabular data file stream and prints the headings

- **void add_datapoint (const Variables &vars, 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 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 and tabular datastream

- **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

- **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

Private Attributes

- Graphics2D * graphics2D
  pointer to the 2D graphics object

- bool win2dOn
  flag to indicate if 2D graphics window is active

- bool tabularDataFlag
  flag to indicate if tabular data stream is active

- int graphicsCntr
  used for x axis values in 2D graphics and for 1st column in tabular data

- std::string tabularCntrLabel
  label for counter used in first line comment w/i the tabular data file

- std::ofstream tabularDataFStream
  file stream for tabulation of graphics data within compute_response

13.47.1 Detailed Description

The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics as well as tabular cataloguing of data for post-processing with Matlab, Tecplot, etc. There is only one Graphics object (dakotaGraphics) and it is global (for convenient access from strategies, models, and approximations).

13.47.2 Member Function Documentation

13.47.2.1 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).
13.47 Graphics Class Reference

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().

13.47.2.2 void create_tabular_datastream (const Variables & vars, const Response & response, const std::string & tabular_data_file)

opens the tabular data file stream and prints the headings Opens the tabular data file stream and prints headings, one for each 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 Graphics::tabularCntrLabel, Graphics::tabularDataFlag, and Graphics::tabularDataFStream.

Referenced by SurrBasedMinimizer::initialize_graphics(), and Iterator::initialize_graphics().

13.47.2.3 void adddatapoint (const Variables & vars, 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(), Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Response::function_values(), Graphics::graphics2D, Graphics::graphicsCntr, Graphics::tabularDataFlag, Graphics::tabularDataFStream, Graphics::win2dOn, and Dakota::write_data_tabular().

Referenced by Model::compute_response(), NonDLocalReliability::mean_value(), SurrBasedLocalMinimizer::minimize_surrogates(), Model::synchronize(), Model::synchronize_nowait(), and NonDLocalReliability::update_level_data().

13.47.2.4 void adddatapoint (int i, double x, double y)

adds data to a single window in the 2d graphics Adds data to a single 2d plot. Allows complete flexibility in defining other kinds of x-y plotting in the 2D graphics.

References Graphics::graphics2D, and Graphics::win2dOn.

13.47.2.5 void newdataset (int i)

creates a separate line graphic for subsequent data points for a single window in the 2d graphics Used for displaying multiple data sets within the same plot.

References Graphics::graphics2D, and Graphics::win2dOn.

Referenced by NonDLocalReliability::update_level_data().

The documentation for this class was generated from the following files:

- DakotaGraphics.hpp
• DakotaGraphics.cpp
13.48 GridApplicInterface Class Reference

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus. Inheritance diagram for GridApplicInterface:

![Inheritance Diagram]

**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)

- 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 \texttt{idSet}
  
  _Set of function evaluation id's for active asynchronous system call evaluations._

- IntShortMap \texttt{failCountMap}
  
  _map linking function evaluation id’s to number of response read failures_

- \texttt{start\_grid\_computing\_t start\_grid\_computing}
  
  _handle to dynamically linked start\_grid\_computing function_

- \texttt{perform\_analysis\_t perform\_analysis}
  
  _handle to dynamically linked perform\_analysis grid function_

- \texttt{get\_jobs\_completed\_t get\_jobs\_completed}
  
  _handle to dynamically linked get\_jobs\_completed grid function_

- \texttt{stop\_grid\_computing\_t stop\_grid\_computing}
  
  _handle to dynamically linked stop\_grid\_computing function_

### 13.48.1 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 \texttt{SysCallApplicInterface} adapted for use with an external grid services library which was dynamically linked using \texttt{dlopen()} services.

### 13.48.2 Member Function Documentation

#### 13.48.2.1 \texttt{void wait\_local\_evaluations (PRPQueue & \_prp\_queue)} [inline, 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 \texttt{SysCallApplicInterface}.

References \texttt{ApplicationInterface::completionSet}, and \texttt{GridApplicInterface::test\_local\_evaluations()}. Referenced by \texttt{GridApplicInterface::derived\_map()}.

#### 13.48.2.2 \texttt{int synchronous\_local\_analysis (int analysis\_id)} [inline, virtual]

This code provides the derived function used by \texttt{ApplicationInterface::serve\_analyses\_synch()}

TODO - allow local analyses?????

Reimplemented from \texttt{SysCallApplicInterface}.
References SysCallApplicInterface::spawn_analysis_to_shell().

The documentation for this class was generated from the following files:

- GridApplicInterface.hpp
- GridApplicInterface.cpp
13.49 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*

Protected Member Functions

- **void derived_compute_response (const ActiveSet &set)**
  
  *portion of compute_response() specific to HierarchSurrModel*

- **void derived_asynch_compute_response (const ActiveSet &set)**
  
  *portion of asynch_compute_response() 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 lowFidelityModel*

- **Model & truth_model ()**
  
  *return highFidelityModel*

- **void derived_subordinate_models (ModelList &ml, bool recurse_flag)**
  
  *return lowFidelityModel and highFidelityModel*
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 highFidelityModel for any lower-level surrogate recursions.

void surrogate_function_indices (const IntSet &surr_fn_indices)
(re)set the surrogate index set in SurrogateModel::surrogateFnIndices

void build_approximation ()
use highFidelityModel to compute the truth values needed for correction of lowFidelityModel results

void component_parallel_mode (short mode)
update component parallel mode for supporting parallelism in lowFidelityModel and highFidelityModel

void derived_init_communicators (int max_iterator_concurrency, bool recurse_flag=true)
set up lowFidelityModel and highFidelityModel for parallel operations

void derived_init_serial ()
set up lowFidelityModel and highFidelityModel for serial operations.

void derived_set_communicators (int max_iterator_concurrency, bool recurse_flag=true)
set active parallel configuration within lowFidelityModel and highFidelityModel

void derived_free_communicators (int max_iterator_concurrency, bool recurse_flag=true)
deallocate communicator partitions for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelityModel)

void serve (int max_iterator_concurrency)
Service lowFidelityModel and highFidelityModel 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 lowFidelityModel and highFidelityModel 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

int evaluation_id () const
Return the current evaluation id for the HierarchSurrModel.

void set_evaluation_reference ()
set the evaluation counter reference points for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelityModel)
void fine_grained_evaluation_counters ()
request fine-grained evaluation reporting within lowFidelityModel and highFidelityModel

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 lowFidelityModel and highFidelityModel)

virtual void eval_tag_prefix (const String &eval_id_str)
set the hierarchical eval ID tag prefix

Private Member Functions

void update_model (Model &model)
update the incoming model (lowFidelityModel or highFidelityModel) with current variable values/bounds/labels

Private Attributes

int hierModelEvalCntr
number of calls to derived_compute_response() / derived_asynch_compute_response()

IntResponseMap cachedTruthRespMap
map of high-fidelity responses retrieved in derived_synchronize_nowait() that could not be returned since corresponding low-fidelity response portions were still pending.

Model lowFidelityModel
provides approximate low fidelity function evaluations. Model is of arbitrary type and supports recursions (e.g., lowFidelityModel can be a data fit surrogate on a low fidelity model).

Model highFidelityModel
provides truth evaluations for computing corrections to the low fidelity results. Model is of arbitrary type and supports recursions.

Response highFidRefResponse
the reference high fidelity response computed in build_approximation() and used for calculating corrections.

String evalTagPrefix
cached evalTag Prefix from parents to use at compute_response time
13.49 HierarchSurrModel Class Reference

13.49.1 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. In particular, it uses a low fidelity model as a surrogate for a high fidelity model. The class contains a lowFidelityModel which performs the approximate low fidelity function evaluations and a highFidelityModel which provides truth evaluations for computing corrections to the low fidelity results.

13.49.2 Member Function Documentation

13.49.2.1 void derived_compute_response (const ActiveSet & set)  [protected, virtual]

portion of compute_response() specific to **HierarchSurrModel** Compute the response synchronously using lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel 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(), DiscrepancyCorrection::apply(), SurrogateModel::approxBuilds, SurrogateModel::asv_mapping(), HierarchSurrModel::build_approximation(), HierarchSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), Model::compute_response(), DiscrepancyCorrection::computed(), Response::copy(), Model::current_response(), Model::currentResponse, Model::currentVariables, SurrogateModel::deltaCorr, Model::eval_tag_prefix(), HierarchSurrModel::evalTagPrefix, SurrogateModel::force_rebuild(), Model::hierarchicalTagging, HierarchSurrModel::highFidelityModel, HierarchSurrModel::highFidRefResponse, HierarchSurrModel::lowFidelityModel, Model::outputLevel, ActiveSet::request_vector(), SurrogateModel::response_mapping(), SurrogateModel::responseMode, Response::update(), and HierarchSurrModel::update_model().

13.49.2.2 void derived_asynch_compute_response (const ActiveSet & set)  [protected, virtual]

portion of asynch_compute_response() specific to **HierarchSurrModel** Compute the response asynchronously using lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel 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 DiscrepancyCorrection::apply(), SurrogateModel::approxBuilds, SurrogateModel::asv_mapping(), Model::asynch_compute_response(), Model::asynch_flag(), HierarchSurrModel::build_approximation(), SurrogateModel::cachedApproxRespMap, HierarchSurrModel::cachedTruthRespMap, HierarchSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), Model::compute_response(), DiscrepancyCorrection::computed(), Response::copy(), Variables::copy(), Model::current_response(), Model::currentVariables, SurrogateModel::deltaCorr, ActiveSet::derivative_vector(), Model::eval_tag_prefix(), HierarchSurrModel::evalTagPrefix, Model::evaluation_id(), SurrogateModel::force_rebuild(), Model::hierarchicalTagging, HierarchSurrModel::highFidRefResponse, HierarchSurrModel::lowFidelityModel, Model::outputLevel, SurrogateModel::rawVarsMap, ActiveSet::request_vector(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::truthIdMap, and HierarchSurrModel::update_model().
13.49.2.3  const IntResponseMap & derived_synchronize () [protected, virtual]

portion of synchronize() specific to HierarchSurrModel Blocking retrieval of asynchronous evaluations from low-
FidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, apply correction (if
active) to each response in the array. derived_synchronize() is designed for the general case where derived_-
asynch_compute_response() may be inconsistent in its use of low fidelity evaluations, high fidelity evaluations, or
both.
Reimplemented from Model.
References DiscrepancyCorrection::apply(), SurrogateModel::cachedApproxRespMap, HierarchSur-
rModel::cachedTruthRespMap, HierarchSurrModel::component_parallel_mode(), DiscrepancyCorrec-
tion::compute(), DiscrepancyCorrection::computed(), SurrogateModel::deltaCorr, HierarchSurrModel::derived_-
synchronize_nowait(), HierarchSurrModel::highFidelityModel, HierarchSurrModel::highFidRefResponse,
HierarchSurrModel::lowFidelityModel, Model::outputLevel, SurrogateModel::rawVarsMap, SurrogateModel::response_mapping(),
SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, Model::synchronize(), and
SurrogateModel::truthIdMap.

13.49.2.4  const IntResponseMap & derived_synchronize_nowait () [protected, virtual]

portion of synchronize_nowait() specific to HierarchSurrModel Nonblocking retrieval of asynchronous evalua-
tions from lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, apply
correction (if active) to each response in the map. derived_synchronize_nowait() is designed for the general case
where derived_asynch_compute_response() may be inconsistent in its use of actual evals, approx evals, or both.
Reimplemented from Model.
References DiscrepancyCorrection::apply(), SurrogateModel::cachedApproxRespMap, HierarchSur-
rModel::cachedTruthRespMap, HierarchSurrModel::component_parallel_mode(), DiscrepancyCorrec-
tion::compute(), DiscrepancyCorrection::computed(), SurrogateModel::deltaCorr, HierarchSurrModel::derived_-
synchronize_nowait(), HierarchSurrModel::highFidelityModel, HierarchSurrModel::highFidRefResponse, HierarchSurrModel::lowFidelityModel,
Model::outputLevel, SurrogateModel::rawVarsMap, SurrogateModel::response_mapping(), Surrogate-
Model::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, Model::synchronize_-
nowait(), and SurrogateModel::truthIdMap.
Referenced by HierarchSurrModel::derived_synchronize().

13.49.2.5  int evaluation_id () const [inline, protected, virtual]

Return the current evaluation id for the HierarchSurrModel. return the hierarchical model evaluation count. Due
to possibly intermittent use of surrogate bypass, this is not the same as either the loFi or hiFi model evaluation
counts. It also does not distinguish duplicate evals.
Reimplemented from Model.
References HierarchSurrModel::hierModelEvalCntr.
The documentation for this class was generated from the following files:

- HierarchSurrModel.hpp
- HierarchSurrModel.cpp
13.50 HybridStrategy Class Reference

Base class for hybrid minimization strategies. Inheritance diagram for HybridStrategy:

```
Strategy

HybridStrategy

CollaborativeHybridStrategy EmbeddedHybridStrategy SequentialHybridStrategy
```

Protected Member Functions

- **HybridStrategy** (ProblemDescDB &problem_db)
  
  *constructor*

- **~HybridStrategy** ()
  
  *destructor*

- void **allocate_methods** ()
  
  *initialize selectedIterators and userDefinedModels*

- void **deallocate_methods** ()
  
  *free communicators for selectedIterators and userDefinedModels*

Protected Attributes

- StringArray **methodList**
  
  *the list of method identifiers*

- int **numIterators**
  
  *number of methods in methodList*

- IteratorArray **selectedIterators**
  
  *the set of iterators, one for each entry in methodList*

- ModelArray **userDefinedModels**
  
  *the set of models, one for each iterator*
13.50.1 Detailed Description

Base class for hybrid minimization strategies. This base class shares code for three approaches to hybrid minimization: (1) the sequential hybrid; (2) the embedded hybrid; and (3) the collaborative hybrid.

The documentation for this class was generated from the following files:

- HybridStrategy.hpp
- HybridStrategy.cpp
13.51 Interface Class Reference

Base class for the interface class hierarchy. Inheritance diagram for Interface::

```
  Interface
   |         |
   v         v
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)**
  
  *copy constructor*

- **virtual ~Interface ()**
  
  *destructor*

- **Interface operator= (const Interface &interface)**
  
  *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 & synch ()**
  
  *recovers data from a series of asynchronous evaluations (blocking)*

- **virtual const IntResponseMap & synch_nowait ()**

Dakota Version 5.4 Developers Manual generated on November 7, 2013
reovers 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_iterator_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_iterator_concurrency)
  set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

• virtual void free_communicators ()
  deallocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

• 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 String 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_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)
  builds the approximation

- virtual void `rebuild_approximation` (const BoolDeque &rebuild_deque)
  rebuilds the approximation after a data update

- virtual void `pop_approximation` (bool save_surr_data)
  removes data from last append from the approximation

- virtual void `restore_approximation` ()
  restores the approximation to a selected previous state

- virtual bool `restore_available` ()
  queries the approximation for the ability to restore a previous increment

- virtual void `finalize_approximation` ()
  finalizes the approximation by applying all trial increments

- virtual void `store_approximation` ()
  move the current approximation into storage for later combination

- virtual void `combine_approximation` (short corr_type)
  combine the current approximation with one previously stored

- 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_saved` ()
  clears saved data (from pop invocations) from an approximation interface
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 ()
  retrieve the approximation coefficients from each Approximation within an ApproximationInterface

virtual const RealVectorArray & approx_coeffs)
  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 const RealVector & approximation_coefficients (const RealVectorArray &approx_coeffs)
  set the approximation coefficients within 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 void file_cleanup () const
  clean up any interface parameter/response files when aborting

void assign_rep (Interface *interface_rep, bool ref_count_incr=true)
  replaces existing letter with a new one

const String & 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

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_flag () const  
  returns a flag signaling the use of multiprocessor evaluation partitions

- **bool** iterator_eval_dedicated_master_flag () 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 async 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

- **String** interfaceType
  
  *the interface type: 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).*

- **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*

- **int** evalIdCntr
  
  *total interface evaluation counter*

- **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 after either a blocking or nonblocking schedule of asynchronous evaluations.

• StringArray `fnLabels`
  response function descriptors from the DAKOTA input file (used in `print_evaluation_summary()` and derived direct interface classes)

• bool `multiProcEvalFlag`
  flag for multiprocessor evaluation partitions (evalComm)

• bool `ieDedMasterFlag`
  flag for dedicated master partitioning at the iterator level

• short `outputLevel`
  output verbosity level: {SILENT, QUIET, NORMAL, VERBOSE, DEBUG}_OUTPUT

• 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
13.51 Interface Class Reference

13.51.1 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.

13.51.2 Constructor & Destructor Documentation

13.51.2.1 Interface ()

default constructor used in Model envelope class instantiations

13.51.2.2 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.

13.51.2.3 Interface (const Interface & interface)

copy constructor Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.
References Interface::interfaceRep, and Interface::referenceCount.

13.51.2.4 ~Interface () [virtual]

destructor Destructor decrements referenceCount and only deletes interfaceRep if referenceCount is zero.
References Interface::interfaceRep, and Interface::referenceCount.

13.51.2.5 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::algebraicFtTags, Interface::algebraicFnTypes, Interface::algebraicMappings, Interface::algebraicVarTags, Interface::asl, Interface::fineGrainEvalCounters, Interface::fnLabels,
ProblemDescDB::get_sa(), ProblemDescDB::get_string(), Interface::init_evaluation_contours(), Interface::outputLevel, and Dakota::strends().

13.51.3 Member Function Documentation

13.51.3.1 Interface operator= (const Interface & interface)


References Interface::interfaceRep, and Interface::referenceCount.

13.51.3.2 void assign_rep (Interface ∗ interface_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 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(), model_interface_plugins(), and run_dakota().

13.51.3.3 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_compute_response(), SingleModel::eval_tag_prefix(), and Interface::eval_tag_prefix().

13.51.3.4 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::synch(), and ApplicationInterface::synch_nowait().

13.51.3.5 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.

References ProblemDescDB::get_string(), and Interface::interface_type().

Referenced by Interface::Interface().

13.51.4 Member Data Documentation

13.51.4.1 IntResponseMap rawResponseMap [protected]

Set of responses returned after either a blocking or nonblocking schedule of asynchronous evaluations. 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(), ApplicationInterface::process_async_local(), ApplicationInterface::process_synch_local(), ApplicationInterface::receive_evaluation(), ApproximationInterface::synch(), ApplicationInterface::synch(), ApproximationInterface::synch_nowait(), ApplicationInterface::synch_nowait(), ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

The documentation for this class was generated from the following files:

- DakotaInterface.hpp
- DakotaInterface.cpp

The documentation for this class was generated from the following files:
13.52 Iterator Class Reference

Base class for the iterator class hierarchy. Inheritance diagram for Iterator::

Public Member Functions

- **Iterator ()**
  
  *default constructor*

- **Iterator (Model &model)**
  
  *standard envelope constructor*

- **Iterator (const String &method_name, Model &model)**
  
  *alternate envelope constructor for instantiations by name*

- **Iterator (const Iterator &iterator)**
  
  *copy constructor*

- **virtual ~Iterator ()**
  
  *destructor*

- **Iterator operator= (const Iterator &iterator)**
  
  *assignment operator*

- **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_iterator (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- **virtual void run ()**
  
  *run portion of run_iterator; 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_iterator (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 **reset** ()
  
  restore initial state for repeated sub-iterator executions

• 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 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 **response_results_active_set** (const ActiveSet &set)
  
  set the requested data for the final iterator response results

• virtual void **initialize_graphics** (bool graph_2d, bool tabular_data, const String &tabular_file)
  
  initialize the 2D graphics window and the tabular graphics data

• virtual void **print_results** (std::ostream &s)
  
  print the final iterator results

• 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 const String & sampling_scheme () const
    return sampling name

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 String 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 bool compact_mode () const
    returns Analyzer::compactMode

void run_iterator (std::ostream &s)
    orchestrate initialize/pre/run/post/finalize phases

void assign_rep (Iterator *iterator_rep, bool ref_count_incr=true)
    replaces existing letter with a new one

ProblemDescDB & problem_description_db () const
    return the problem description database (probDescDB)

const String & method_name () const
    return the method name

const String & method_id () const
    return the method identifier (methodId)

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.

• int `maximum_concurrency () const`
  return the maximum concurrency supported by the iterator

• void `maximum_concurrency` (int max_conc)
  set the maximum concurrency supported by the iterator

• 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 &dr_index1, const ShortArray &c_target2, const ShortArray &di_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
Protected Member Functions

- **Iterator (BaseConstructor, Model &model)**
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Iterator (NoDBBaseConstructor, Model &model)**
  
  alternate constructor for base iterator classes constructed on the fly

- **Iterator (NoDBBaseConstructor)**
  
  alternate constructor for base iterator classes constructed on the fly

- **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

Protected Attributes

- **Model iteratedModel**
  
  shallow copy of the model passed into the constructor or a thin RecastModel wrapped around it

- **ProblemDescDB & probDescDB**
  
  class member reference to the problem description database

- **String 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 maxConcurrency**
  
  maximum coarse-grained concurrency

- **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 numDiscreteRealVars
  number of active discrete real vars

- size_t numFinalSolutions
  number of solutions to retain in best variables/response arrays

- ActiveSet activeSet
  tracks the response data requirements on each function evaluation

- 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 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 secondaryADRVarMapTargets
  "secondary" all discrete real variable mapping targets flowed down from higher level iteration

- 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*

• Real `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*

• Real `fdHessByGradStepSize`
  *relative finite difference step size for numerical Hessians estimated using first-order differences of gradients*

• Real `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*

• 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*

• bool `asynchFlag`
  *copy of the model’s asynchronous evaluation flag*

• int `writePrecision`
  *write precision as specified by the user*

• ResultsManager & `resultsDB`
  *reference to the global iterator results database*

• ResultsNames `resultsNames`
  *valid names for iterator results*
Private Member Functions

- **Iterator ∗ get_iterator (Model &model)**
  
  Used by the envelope to instantiate the correct letter class.

- **Iterator ∗ get_iterator (const String &method_name, Model &model)**
  
  Used by the envelope to instantiate the correct letter class.

- **virtual void pre_output ()**
  
  convenience function to write variables to file, following pre-run

- **virtual void post_input ()**
  
  read tabular data for post-run mode

Private Attributes

- **String methodId**
  
  method identifier string from the input file

- **Iterator ∗ iteratorRep**
  
  pointer to the letter (initialized only for the envelope)

- **int referenceCount**
  
  number of objects sharing iteratorRep

- **size_t execNum**
  
  an execution number for this instance of the class, unique across all instances of same methodName/methodId

### 13.52.1 Detailed Description

Base class for the iterator class hierarchy. The Iteration 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.

### 13.52.2 Constructor & Destructor Documentation

#### 13.52.2.1 Iterator ()

default constructor The default constructor is used in Vector<Iterator> instantiations and for initialization of Iterator objects contained in Strategy derived classes (see derived class header files). iteratorRep is NULL in this case (a populated problem_db is needed to build a meaningful Iterator object). This makes it necessary to check for NULL pointers in the copy constructor, assignment operator, and destructor.
Referenced by SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), and SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

13.52.2.2 Iterator (Model & model)

standard envelope constructor Used in iterator instantiations within strategy constructors. Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

13.52.2.3 Iterator (const String & methodName, Model & model)

alternate envelope constructor for instantiations by name 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. References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

13.52.2.4 Iterator (const Iterator & iterator)


13.52.2.5 ~Iterator () [virtual]

destructor Destructor decrements referenceCount and only deletes iteratorRep when referenceCount reaches zero. References Iterator::iteratorRep, and Iterator::referenceCount.

13.52.2.6 Iterator (BaseConstructor, Model & model) [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 builds the base class data for all inherited 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 Dakota::abort_handler(), Iterator::fdGradStepSize, Iterator::fdHessByFnStepSize, Iterator::fdHessByGradStepSize, ProblemDescDB::get_is(), ProblemDescDB::get_rv(), Iterator::gradientType, Iterator::hessianType, Iterator::intervalType, Iterator::methodName, Iterator::methodSource, Iterator::numContinuousVars, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, Iterator::numFunctions, and Iterator::probDescDB.
13.52.2.7 Iterator (NoDBBaseConstructor, Model & model) [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. Therefore it only sets attributes taken from the incoming model. Since there are no iterator-specific redefinitions of maxIterations or numFinalSolutions in NoDBBaseConstructor mode, go ahead and assign default value for all iterators.

13.52.2.8 Iterator (NoDBBaseConstructor) [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. 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. Since there are no iterator-specific redefinitions of maxIterations or numFinalSolutions in NoDBBaseConstructor mode, go ahead and assign default value for all iterators.

13.52.3 Member Function Documentation

13.52.3.1 Iterator operator= (const Iterator & iterator)


References Iterator::iteratorRep, and Iterator::referenceCount.

13.52.3.2 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 CONMINOptimizer, LeastSq, Minimizer, NonD, Optimizer, DOTOptimizer, NLPQLPOptimizer, SNLLLeastSq, and SNLLOptimizer.

References Model::asynch_flag(), Iterator::asynchFlag, Iterator::initialize_run(), Model::is_null(), Iterator::iteratedModel, Iterator::iteratorRep, Iterator::maxConcurrency, Model::set_communicators(), Model::set_evaluaton_reference(), and Iterator::summaryOutputFlag.

Referenced by Iterator::initialize_run(), and Iterator::run_iterator().

13.52.3.3 void pre_run () [virtual]

pre-run portion of run_iterator (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 DDACEDesignCompExp, FSUDesignCompExp, NonDLHSSampling, ParamStudy, and PSUADEDesignCompExp.

References Iterator::iteratorRep, and Iterator::pre_run().

Referenced by Iterator::pre_run(), and Iterator::run_iterator().

13.52.3.4 void run () [virtual]

run portion of run_iterator; 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 in LeastSq, NonD, Optimizer, PStudyDACE, Verification, and SurrBasedMinimizer.

References Dakota::abort_handler(), Iterator::iteratorRep, and Iterator::run().

Referenced by Iterator::run(), and Iterator::run_iterator().

13.52.3.5 void post_run (std::ostream & s) [virtual]

post-run portion of run_iterator (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 COLINOptimizer, LeastSq, Optimizer, DDACEDesignCompExp, FSUDesignCompExp, NonDLHSSampling, ParamStudy, PSUADEDesignCompExp, SNLLLeastSq, and SNLLOptimizer.

References Model::is_null(), Iterator::iteratedModel, Iterator::iteratorRep, Iterator::post_run(), Model::print_evaluation_summary(), Iterator::print_results(), Iterator::resultsDB, Iterator::summaryOutputFlag, and ResultsManager::write_databases().

Referenced by Iterator::post_run(), and Iterator::run_iterator().

13.52.3.6 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 LeastSq, Minimizer, NonD, Optimizer, SNLLLeastSq, and SNLLOptimizer.

References Iterator::finalize_run(), and Iterator::iteratorRep.

Referenced by Iterator::finalize_run(), and Iterator::run_iterator().

13.52.3.7 void initialize_graphics (bool graph_2d, bool tabular_data, const String & tabular_file) [virtual]

initialize the 2D graphics window and the tabular graphics data This is a convenience function for encapsulating graphics initialization operations. It does not require a strategyRep forward since it is only used by letter objects.
Reimplemented in NonDReliability, and SurrBasedMinimizer.

References Model::auto_graphics(), Graphics::create_plots_2d(), Graphics::create_tabular_datastream(), Model::current_response(), Model::current_variables(), Dakota::dakota_graphics, Iterator::initialize_graphics(), Iterator::iteratedModel, and Iterator::iteratorRep.

Referenced by Iterator::initialize_graphics(), SequentialHybridStrategy::run_sequential(), SingleMethodStrategy::run_strategy(), EmbeddedHybridStrategy::run_strategy(), ConcurrentStrategy::run_strategy(), and CollaborativeHybridStrategy::run_strategy().

13.52.3.8 void print_results (std::ostream & s) [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 Analyzer, LeastSq, Optimizer, PStudyDACE, Verification, NonDAdaptImpSampling, NonDAdaptiveSampling, NonDExpansion, NonDGlobalReliability, NonDGPImpSampling, NonDIncremLHSSampling, NonDInterval, NonDLHSSampling, NonDLocalReliability, NonDPOFDarts, RichExtrapVerification, and SurrBasedMinimizer.

References Iterator::iteratorRep, and Iterator::print_results().

Referenced by Iterator::post_run(), Iterator::print_results(), and EfficientSubspaceMethod::reduced_space_uq().

13.52.3.9 int num_samples () const [virtual]

get the current number of samples 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 maxConcurrency. May be (is) overridden by derived classes.

Reimplemented in DDACEDesignCompExp, FSUDesignCompExp, NonDCubature, NonDQuadrature, NonDSampling, NonDSparseGrid, and PSUADEDesignCompExp.

References Model::derivative_concurrency(), Iterator::iteratedModel, Iterator::iteratorRep, Iterator::maxConcurrency, and Iterator::num_samples().

Referenced by DataFitSurrModel::build_global(), NonDGlobalReliability::get_best_sample(), Iterator::num_samples(), Analyzer::samples_to_variables_array(), and Analyzer::variables_array_to_samples().

13.52.3.10 void run_iterator (std::ostream & s)

orchestrate initialize/pre/run/post/finalize phases Iterator supports a construct/initialize-run/pre-run/run/post-run/finalize-run/destroy progression. This member (non-virtual) function sequences these run phases; it accepts an ostream, but controls verbosity with outputLevel

References ParallelLibrary::command_line_post_run(), ParallelLibrary::command_line_pre_run(), ParallelLibrary::command_line_run(), Iterator::execNum, Iterator::finalize_run(), ResultsID::increment_id(), Iterator::initialize_run(), ResultsID::instance(), Iterator::iteratedModel, Iterator::iteratorRep, Iterator::method_id(), Iterator::method_name(), Iterator::methodName, Iterator::outputLevel, Model::parallel_library(), Iterator::post_input(), Iterator::post_run(), Iterator::pre_output(), Iterator::pre_run(), Iterator::run(), Iterator::run_iterator(), and Iterator::summaryOutputFlag.
13.52.3.11  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 iteratorRep 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::iteratorRep, and Iterator::referenceCount.

Referenced by NonDExpansion::construct_cubature(), NonDExpansion::construct_expansion_sampler(), NonDAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), EffGlobalMinimizer::EffGlobalMinimizer(), EfficientSubspaceMethod::init_fullspace_sampler(), NonDLocalReliability::method_recourse(), NonDLocalInterval::method_recourse(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDGPImpSampling::NonDGPImpSampling(), NonDGPMASBayesCalibration::NonDGPMASBayesCalibration(), NonDLHSInterval::NonDLHSInterval(), NonDLocalInterval::NonDLocalInterval(), GaussProcApproximation::optimize_theta_global(), GaussProcApproximation::optimize_theta_multipoint(), and SurrBasedLocalMinimizer::relax_constraints().

13.52.3.12  void eval_tag_prefix (const String & eval_id_str) [virtual]

set the hierarchical eval ID tag prefix. This preprend may need to become a virtual function if the tagging should propagate to other subModels or helper Iterators an Iterator may contain.
13.52 Iterator Class Reference

References Model::eval_tag_prefix(), Iterator::eval_tag_prefix(), Iterator::iteratedModel, and Iterator::iteratorRep.

Referenced by DataFitSurrModel::build_global(), NestedModel::derived_compute_response(), and Iterator::eval_tag_prefix().

13.52.3.13 Iterator * get_iterator (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 methodName attribute.

References ProblemDescDB::get_string(), Iterator::method_name(), Iterator::methodName, Iterator::probDescDB, Dakota::strbegins(), and Dakota::strends().

Referenced by Iterator::Iterator().

13.52.3.14 Iterator * get_iterator (const String & method_name, 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_name.

References Dakota::strbegins(), and Dakota::strends().

13.52.4 Member Data Documentation

13.52.4.1 Real 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 DOTOptimizer::initialize(), CONMINOptimizer::initialize(), Iterator::Iterator(), NLSSOLLeastSq::NLSSOLLeastSq(), NPSOLOptimizer::NPSOLOptimizer(), SNLLLeastSq::SNLLLeastSq(), and SNLLOptimizer::SNLLOptimizer().

13.52.4.2 Real fdHessByGradStepSize [protected]

relative finite difference step size for numerical Hessians estimated using first-order differences of gradients A scalar value (instead of the vector fd_hessian_step_size spec) is used within the iterator hierarchy since this attribute is only used to publish a step size to vendor numerical Hessian algorithms.

Referenced by Iterator::Iterator().

13.52.4.3 Real fdHessByFnStepSize [protected]

relative finite difference step size for numerical Hessians estimated using second-order differences of function values A scalar value (instead of the vector fd_hessian_step_size spec) is used within the iterator hierarchy since this attribute is only used to publish a step size to vendor numerical Hessian algorithms.
Referenced by Iterator::Iterator().

The documentation for this class was generated from the following files:

- DakotaIterator.hpp
- DakotaIterator.cpp
13.53  JEGAOptimizer Class Reference

A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA). Inheritance diagram for JEGAOptimizer::

```
Diagram:

  JEGAOptimizer
     |        
     v        
  Optimizer
     |        
     v        
  Minimizer
     |        
     v        
  Iterator
```

**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 find_optimum ()**
  
  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 (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, 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, 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, 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.

Static Private Attributes

• static const std::string SOGA_METHOD_TXT
  
  The text that indicates the SOGA method.

• static const std::string MOGA_METHOD_TXT
  
  The text that indicates the MOGA method.

13.53.1 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).

13.53.2 Constructor & Destructor Documentation

13.53.2.1 JEGAOptimizer (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:

  model The Dakota::Model that will be used by this optimizer for problem information, etc.

References JEGAOptimizer::_theEvalCreator, ProblemDescDB::get_int(), ProblemDescDB::get_short(), Model::init_communicators(), Iterator::iteratedModel, JEGAOptimizer::LoadTheParameterDatabase(), Iterator::maxConcurrency, Iterator::methodName, Minimizer::minimizerRecasts, JEGAOptimizer::MOGA_-METHOD_TXT, Iterator::numFinalSolutions, and Iterator::probDescDB.
13.53.3 Member Function Documentation

13.53.3.1 void LoadDakotaResponses (const JEGA::Utilities::Design & \textit{from}, Dakota::Variables & \textit{vars}, Dakota::Response & \textit{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:**

- \textit{from} The JEGA Design class object from which to extract the variable and response information for Dakota.
- \textit{vars} The Dakota::Variables object into which to load the design variable values of \textit{from}.
- \textit{resp} The Dakota::Response object into which to load the objective function and constraint values of \textit{from}.

References Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Response::function_values(), Iterator::numContinuousVars, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, Iterator::numFunctions, Minimizer::numNonlinearConstraints, and Optimizer::numObjectiveFns.

Referenced by JEGAOptimizer::find_optimum().

13.53.3.2 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.

References JEGAOptimizer::_theParamDB, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_sizet(), ProblemDescDB::get_string(), Iterator::methodName, JEGAOptimizer::MOGA_METHOD_TXT, Iterator::probDescDB, JEGAOptimizer::ReCreateTheParameterDatabase(), and JEGAOptimizer::SOGA_METHOD_TXT.

Referenced by JEGAOptimizer::JEGAOptimizer().

13.53.3.3 void LoadAlgorithmConfig (JEGA::FrontEnd::AlgorithmConfig & \textit{aConfig}) [protected]

Completely initializes the supplied algorithm configuration. This loads the supplied configuration object with appropriate data retrieved from the parameter database.

**Parameters:**

- \textit{aConfig} The algorithm configuration object to load.

References Iterator::method_id(), Iterator::methodName, JEGAOptimizer::MOGA_METHOD_TXT, and JEGAOptimizer::SOGA_METHOD_TXT.

Referenced by JEGAOptimizer::find_optimum().
13.53.3.4 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:

  pConfig  The problem configuration object to load.

References JEGAOptimizer::LoadTheConstraints(), JEGAOptimizer::LoadTheDesignVariables(), and JEGAOptimizer::LoadTheObjectiveFunctions().
Referenced by JEGAOptimizer::find_optimum().

13.53.3.5 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 Iterator::iteratedModel, Iterator::numContinuousVars, Iterator::numDiscreteIntVars, and Iterator::numDiscreteRealVars.
Referenced by JEGAOptimizer::LoadProblemConfig().

13.53.3.6 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.

References Iterator::iteratedModel, Optimizer::numObjectiveFns, Model::primary_response_fn_sense(), and Model::response_labels().
Referenced by JEGAOptimizer::LoadProblemConfig().

13.53.3.7 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(), Iterator::iteratedModel, Minimizer::numLinearEqConstraints, Minimizer::numLinearIneqConstraints, Minimizer::numNonlinearEqConstraints, and Minimizer::numNonlinearIneqConstraints.

Referenced by JEGAOptimizer::LoadProblemConfig().

13.53.3.8 void GetBestSolutions (const JEGA::Utilities::DesignOFSortSet & from, 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.
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.

References JEGAOptimizer::GetBestMOsolutions(), JEGAOptimizer::GetBestSOSolutions(), Iterator::methodName, JEGAOptimizer::MOGA_METHOD_TXT, and JEGAOptimizer::SOGA_METHOD_TXT.

Referenced by JEGAOptimizer::find_optimum().

13.53.3.9 void GetBestMOsolutions (const JEGA::Utilities::DesignOFSortSet & from, 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.

References Iterator::numFinalSolutions.

Referenced by JEGAOptimizer::GetBestSolutions().

13.53.3.10 void GetBestSOSolutions (const JEGA::Utilities::DesignOFSortSet & from, 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 JEGAOptimizer::_theParamDB, and Iterator::numFinalSolutions.

Referenced by JEGAOptimizer::GetBestSolutions().

13.53.3.11 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.

Referenced by JEGAOptimizer::find_optimum().

13.53.3.12 void find_optimum () [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::ExecuteAlgorithm 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.

Implements Optimizer.

References JEGAOptimizer::_initPts, JEGAOptimizer::_theEvalCreator, JEGAOptimizer::_theParamDB, Driver::DestroyAlgorithm(), Driver::ExtractAllData(), JEGAOptimizer::GetBestSolutions(), JEGAOptimizer::initial_points(), JEGAOptimizer::LoadAlgorithmConfig(), JEGAOptimizer::LoadDakotaResponses(), JEGAOptimizer::LoadProblemConfig(), Driver::PerformIterations(), Minimizer::resize_best_resp_array(), Minimizer::resize_best_vars_array(), and JEGAOptimizer::ToDoubleMatrix().

13.53.3.13 bool accepts_multiple_points () const [virtual]

Overridden to return true since JEGA algorithms can accept multiple initial points.

Returns:

true, always.

Reimplemented from Iterator.

13.53.3.14 bool returns_multiple_points () const [virtual]

Overridden to return true since JEGA algorithms can return multiple final points.

Returns:

true, always.

Reimplemented from Iterator.
13.53.3.15  void initial_points (const VariablesArray & pts) [virtual]

Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables.

Parameters:

   pts  The array of initial points for the JEGA algorithm created and run by this JEGAOptimizer.

Reimplemented from Iterator.
References JEGAOptimizer::_initPts.

13.53.3.16  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.
References JEGAOptimizer::_initPts.
Referenced by JEGAOptimizer::find_optimum().

13.53.4  Member Data Documentation

13.53.4.1  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.

Referenced by JEGAOptimizer::find_optimum(), and JEGAOptimizer::initial_points().

The documentation for this class was generated from the following files:

   •  JEGAOptimizer.hpp
   •  JEGAOptimizer.cpp
13.54 LeastSq Class Reference

Base class for the nonlinear least squares branch of the iterator hierarchy. Inheritance diagram for LeastSq:

```
Base class for the nonlinear least squares branch of the iterator hierarchy. Inheritance diagram for LeastSq:

```

Protected Member Functions

- **LeastSq()**
  *default constructor*

- **LeastSq(Model &model)**
  *standard constructor*

- **LeastSq(NoDBBaseConstructor, Model &model)**
  *alternate constructor*

- **~LeastSq()**
  *destructor*

- **void initialize_run()**
- **void run()**
  *run portion of run_iterator; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

- **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)**
- **virtual void minimize_residuals() = 0**
  *Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function for the least squares branch.*

- **void get_confidence_intervals()**
  *Calculate confidence intervals on estimated parameters.*
Protected Attributes

- int 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

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.

13.54.1 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.
13.54 Class Reference

13.54.2 Constructor & Destructor Documentation

13.54.2.1 LeastSq (Model & model) [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, Variables::copy(), Model::current_variables(), Minimizer::data_transform_model(), Model::init_communicators(), Iterator::iteratedModel, Iterator::maxConcurrency, Minimizer::minimizerRecasts, Minimizer::numIterPrimaryFns, LeastSq::numLeastSqTerms, Minimizer::numRowsExpData, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, Minimizer::optimizationFlag, Minimizer::scale_model(), Minimizer::scaleFlag, LeastSq::weight_model(), and LeastSq::weightFlag.

13.54.3 Member Function Documentation

13.54.3.1 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 Minimizer.
Reimplemented in SNLLLeastSq.
References Iterator::iteratedModel, LeastSq::leastSqInstance, Minimizer::obsDataFlag, LeastSq::prevLSqInstance, Minimizer::scaleFlag, and Model::update_from_subordinate_model().

13.54.3.2 void run () [inline, protected, virtual]

run portion of run_iterator; 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 LeastSq::minimize_residuals().

13.54.3.3 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.
Reimplemented in SNLLLeastSq.
References Dakota::abort_handler(), Response::active_set_request_vector(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Variables::continuous_variables(), Response::copy(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Response::function_value(), Response::function_values(), Iterator::iteratedModel, Minimizer::modify_s2n(), Minimizer::need_resp_trans_byvars(), LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Model::primary_response_fn_weights(),
Minimizer::primaryRespScaleFlag, Minimizer::response_modify_s2n(), Minimizer::secondaryRespScaleFlag, Model::subordinate_model(), Response::update_partial(), Minimizer::varsScaleFlag, and LeastSq::weightFlag.

### 13.54.3.4 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.

Reimplemented in SNLLLeastSq.

References LeastSq::leastSqInstance, and LeastSq::prevLSqInstance.

### 13.54.3.5 void print_results (std::ostream & s) [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, LeastSq::confBoundsLower, LeastSq::confBoundsUpper, Model::continuous_variable_labels(), Dakota::data_pairs, Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Iterator::numContinuousVars, Iterator::numFunctions, LeastSq::numLeastSqTerms, Model::primary_response_fn_weights(), ActiveSet::request_values(), Model::subordinate_model(), Dakota::write_data_partial(), and Dakota::write_precision.

### 13.54.3.6 void get_confidence_intervals () [protected]

Calculate confidence intervals on estimated parameters. Calculate individual confidence intervals for each parameter. These bounds are based on a linear approximation of the nonlinear model.

References Iterator::activeSet, Iterator::bestResponseArray, Iterator::bestVariablesArray, Model::compute_response(), LeastSq::confBoundsLower, LeastSq::confBoundsUpper, Model::continuous_variables(), Model::current_response(), Response::function_gradients(), Iterator::iteratedModel, Iterator::numContinuousVars, LeastSq::numLeastSqTerms, ActiveSet::request_values(), Minimizer::scaleFlag, and Minimizer::vendorNumericalGradFlag.

Referenced by NLSSOLLeastSq::minimize_residuals(), NL2SOLLeastSq::minimize_residuals(), and SNLLLeastSq::post_run().

### 13.54.3.7 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 Model::assign_rep(), Iterator::iteratedModel, Iterator::numContinuousVars, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints,
13.54 LeastSq Class Reference

Iterator::outputLevel, LeastSq::primary_resp_weighter(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Minimizer::secondary_resp_copier(), and Model::subordinate_model().

Referenced by LeastSq::LeastSq().

13.54.3.8 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_derivative_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
13.55 MatlabInterface Class Reference

Inheritance diagram for MatlabInterface::

```
  Interface
   
  ApplicationInterface
   
  DirectApplicInterface
   
  MatlabInterface
```

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*
13.55 MatlabInterface Class Reference

13.55.1 Detailed Description

Specialization of DirectApplicInterface to link to Matlab analysis drivers. Includes convenience functions to map data to/from Matlab.

13.55.2 Member Function Documentation

13.55.2.1 int derived_map_ac (const String & ac_name)  [protected, virtual]

execute an analysis code portion of a direct evaluation invocation Matlab specialization of dervied analysis components.
Reimplemented from DirectApplicInterface.
References ApplicationInterface::analysisServerId, and MatlabInterface::matlab_engine_run().

13.55.2.2 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 MATLAB 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::numADR, Dakota::NUMBER_OF_FIELDS, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, DirectApplicInterface::numVars, Interface::outputLevel, DirectApplicInterface::xC, 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
13.56 Minimizer Class Reference

Base class for the optimizer and least squares branches of the iterator hierarchy. Inheritance diagram for Minimizer:

![Inheritance Diagram for Minimizer]

**Public Member Functions**

- void **constraint_tolerance** (Real constr_tol)
  
  set the method constraint tolerance (constraintTol)

- Real **constraint_tolerance** () const
  
  return the method constraint tolerance (constraintTol)

**Protected Member Functions**

- **Minimizer ()**
  
  default constructor

- **Minimizer (Model &model)**
  
  standard constructor

- **Minimizer (NoDBBaseConstructor, Model &model)**
  
  alternate constructor for "on the fly" instantiations

- **Minimizer (NoDBBaseConstructor, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq)**
alternate constructor for "on the fly" instantiations

- \texttt{\~Minimizer()}  
  destructor

- \texttt{void initialize\_run()}  
  utility function to perform common operations prior to \texttt{pre\_run()}; typically memory initialization; setting of instance pointers

- \texttt{void finalize\_run()}  
  utility function to perform common operations following \texttt{post\_run()}; deallocation and resetting of instance pointers

- \texttt{bool data\_transform\_model(bool weight\_flag=false)}  
  Wrap \texttt{iteratedModel} in a \texttt{RecastModel} that subtracts provided observed data from the primary response functions (variables and secondary responses are unchanged).

- \texttt{void scale\_model()}  
  Wrap \texttt{iteratedModel} in a \texttt{RecastModel} that performs variable and/or response scaling.

- \texttt{bool need\_resp\_trans\_byvars(const ShortArray \&asv, int start\_index, int num\_resp)}  
  determine if response transformation is needed due to variable transformations

- \texttt{RealVector modify\_s2n(const RealVector \&scaled\_vars, constIntArray \&scale\_types, const RealVector \&multipliers, const RealVector \&offsets) const}  
  general \texttt{RealVector} mapping from scaled to native variables (and values)

- \texttt{void response\_modify\_s2n(const Variables \&native\_vars, const Response \&scaled\_response, Response \&native\_response, int start\_offset, int num\_responses) const}  
  map responses from scaled to native space

- \texttt{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

- \texttt{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 \texttt{userPrimaryFns}

- \texttt{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

- \texttt{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.*

**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*

• static void **replicate_set_recast** (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  
  *conversion of request vector values for Least Squares*

• static void **secondary_resp_copier** (const Variables &input_vars, const Variables &output_vars, const Response &input_response, Response &output_response)
  
  *copy the partial response for secondary functions when needed (data and reduction transforms)*

**Protected Attributes**

• 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

- int numNonlinearConstraints
  total number of nonlinear constraints

- int numLinearConstraints
  total number of linear constraints

- int numConstraints
  total number of linear and nonlinear constraints

- unsigned short minimizerRecasts
  number of RecastModels locally (in Minimizer or derived classes) layered over the initially passed in Model

- 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 user’s model always initialize at Minimizer, even if overridden later

- size_t numIterPrimaryFns
  number of objective functions or least squares terms in iterator’s view 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
- **String obsDataFilename**
  
  filename from which to read observed data

- **bool obsDataFlag**
  
  flag indicating whether user-supplied 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

- **IntVector numReplicates**
  
  number of replicates

- **size_t numRowsExpData**
  
  number of total rows of data since we are allowing varying numbers of experiments and replicates per experiment

- **bool scaleFlag**
  
  flag for overall scaling status

- **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

• Minimizer * `prevMinInstance`
  pointer containing previous value of minimizerInstance

• bool `vendorNumericalGradFlag`
  convenience flag for gradType == numerical && methodSource == vendor

**Static Protected Attributes**

• static Minimizer * `minimizerInstance`
  pointer to Minimizer used in static member functions

**Private Member Functions**

• bool `data_difference_core` (const Response &raw_response, Response &residual_response)
  Core of data difference, which doesn’t perform any output.

• void `initialize_scaling` ()
  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.

- `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 response_scaler_core` (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &iterator_response, size_t start_offset, size_t num_responses)
  Core of response scaling, which doesn’t perform any output.

- `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:

- `void response_modify_n2s` (const Variables &scaled_vars, const Response &native_response, Response &scaled_response, int start_offset, int num_responses) const
  map responses from native to scaled variable space

- `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

- `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

**Static Private Member Functions**

- `static void primary_resp_differencer` (const Variables &raw_vars, const Variables &residual_vars, const Response &raw_response, Response &residual_response)
  Recast callback function to difference residuals with observed data.

- `static void variables_scaler` (const Variables &scaled_vars, Variables &native_vars)
  RecastModel callback for variables scaling: transform variables from scaled to native (user) 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.

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)

13.56.1 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.

13.56.2 Constructor & Destructor Documentation

13.56.2.1 Minimizer (Model & model) [protected]

standard constructor This constructor extracts inherited data for the optimizer and least squares branches and performs sanity checking on constraint settings.

References Dakota::abort_handler(), Response::active_set_request_vector(), Iterator::bestResponseArray, Minimizer::bigIntBoundSize, Minimizer::bigRealBoundSize, Minimizer::boundConstraintFlag, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Response::copy(), Model::current_response(), Model::discrete_int_lower_bounds(), Model::discrete_int_upper_bounds(), Model::discrete_real_lower_bounds(), Model::discrete_real_upper_bounds(), Iterator::gradientType, Iterator::hessianType, Iterator::iteratedModel, Iterator::maxIterations, Iterator::methodName, Iterator::methodSource, Iterator::numContinuousVars, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, Iterator::numFinalSolutions, Iterator::numFunctions, Minimizer::numLinearEqConstraints, Minimizer::numLinearIneqConstraints, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Dakota::strbegins(), Dakota::strends(), and Minimizer::vendorNumericalGradFlag.

13.56.3 Member Function Documentation

13.56.3.1 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 CONMINOptimizer, LeastSq, Optimizer, DOTOptimizer, NLPQLPOptimizer, SNLLLeastSq, and SNLLOptimizer.

References Model::all_continuous_variables(), Model::all_discrete_int_variables(), Model::all_discrete_real_variables(), Iterator::bestVariablesArray, Iterator::iteratedModel, Minimizer::minimizerInstance, Minimizer::minimizerRecasts, Minimizer::prevMinInstance, Iterator::subIteratorFlag, and Model::subordinate_model().

13.56.3.2 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.
Reimplemented in LeastSq, Optimizer, SNLLLeastSq, and SNLLOptimizer.
References Minimizer::minimizerInstance, and Minimizer::prevMinInstance.

13.56.3.3 bool data_transform_model (bool weight_flag = false) [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 userDefinedModel. This will set weights to sigma[i]**-2 if appropriate. weight_flag is true is there already exist user-specified weights in the calling context.
References Dakota::abort_handler(), Iterator::activeSet, Model::assign_rep(), Minimizer::expData, ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_sizet(), Iterator::iteratedModel, ExperimentData::load_scalar(), Iterator::numContinuousVars, Minimizer::numExperiments, Iterator::numFunctions, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numReplicates, Minimizer::numRowsExpData, Minimizer::numUserPrimaryFns, Minimizer::obsDataFilename, Iterator::outputLevel, Minimizer::primary_resp_differencer(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Iterator::probDescDB, Minimizer::replicate_set_recast(), ActiveSet::request_vector(), Minimizer::secondary_resp_copier(), and Model::subordinate_model().
Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

13.56.3.4 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(), Minimizer::cvScaleTypes, RecastModel::initialize(), Minimizer::initialize_scaling(), Iterator::iteratedModel, Model::model_rep(), Iterator::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Iterator::outputLevel, Minimizer::primary_resp_scaler(), Model::primary_response_fn_sense(),
13.56 Minimizer Class Reference

Model::primary_response_fn_weights(), Minimizer::primaryRespScaleFlag, Minimizer::responseScaleTypes, Minimizer::secondary_resp_scaler(), Minimizer::secondaryRespScaleFlag, Model::subordinate_model(), Minimizer::variables_scaler(), and Minimizer::varsScaleFlag.

Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

13.56.3.5  void gnewton_set_recast (const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set)  [static, protected]

collection of request vector values for the Gauss-Newton Hessian approximation For Gauss-Newton Hessian requests, activate the 2 bit and mask the 4 bit.

References ActiveSet::request_value(), and ActiveSet::request_vector().

Referenced by Optimizer::reduce_model(), and SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

13.56.3.6  void secondary_resp_copier (const Variables & input_vars, const Variables & output_vars, const Response & input_response, Response & output_response)  [static, protected]

copy the partial response for secondary functions when needed (data and reduction transforms) Constraint function
map from user/native space to iterator/scaled/combined space using a RecastModel.

References Minimizer::minimizerInstance, Minimizer::numIterPrimaryFns, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, and Response::update_partial().

Referenced by Minimizer::data_transform_model(), Optimizer::reduce_model(), and LeastSq::weight_model().

13.56.3.7  bool need_resp_trans_byvars (const ShortArray & asv, int start_index, int num_resp)  [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 Minimizer::varsScaleFlag.

Referenced by SNLLLeastSq::post_run(), Optimizer::post_run(), LeastSq::post_run(), and Minimizer::response_scaler_core().

13.56.3.8  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 SNLLLeastSq::post_run(), Optimizer::post_run(), LeastSq::post_run(), and Minimizer::variables_scaler().
13.56.3.9  void response_modify_s2n (const Variables & native_vars, const Response & scaled_response, Response & native_response, int start_offset, int num_responses) const [protected]

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

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(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_labels(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Minimizer::numUserPrimaryFns, Iterator::outputLevel, Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, Dakota::write_col_vector_trans(), Dakota::write_data(), and Dakota::write_precision.

Referenced by Optimizer::post_run(), and LeastSq::post_run().

13.56.3.10  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::lagrangian_merit(), Optimizer::objective_reduction(), SurrBasedMinimizer::penalty_merit(), COLINOptimizer::post_run(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

13.56.3.11  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.

13.56.3.12  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 nonlinear-RespMapping settings.

References Iterator::numContinuousVars, and Minimizer::optimizationFlag.

13.56.3.13  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(), Iterator::numContinuousVars, and Minimizer::optimizationFlag.

13.56.3.14  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(), Iterator::iteratedModel, Minimizer::minimizerRecasts, and Model::subordinate_model().

Referenced by JEGAOptimizer::find_optimum(), and COLINOptimizer::post_run().

13.56.3.15  void resize_best_resp_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(), Iterator::iteratedModel, Minimizer::minimizerRecasts, and Model::subordinate_model().

Referenced by JEGAOptimizer::find_optimum(), and COLINOptimizer::post_run().

13.56.3.16  void primary_resp_differencer (const Variables & raw_vars, const Variables & residual_vars, const Response & raw_response, Response & residual_response)  [static, private]

Recast callback function to difference residuals with observed data. Difference the primary responses with observed data.
References Minimizer::data_difference_core(), Response::function_labels(), Response::function_values(), Minimizer::minimizerInstance, Iterator::outputLevel, and Dakota::write_data().

Referenced by Minimizer::data_transform_model().

13.56.3.17  void initialize_scaling () [private]

initialize scaling types, multipliers, and offsets; perform error checking. Initialize scaling types, multipliers, and offsets. Update the iteratedModel appropriately.

References Dakota::abort_handler(), Minimizer::compute_scaling(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variable_labels(), Model::continuous_variables(), Dakota::copy_data(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, ProblemDescDB::get_rv(), ProblemDescDB::get_sa(), Iterator::iteratedModel, Minimizer::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(), Minimizer::linearEqScaleMultipliers, Minimizer::linearEqScaleOffsets, Minimizer::linearEqScaleTypes, Minimizer::linearIneqScaleMultipliers, Minimizer::linearIneqScaleOffsets, Minimizer::linearIneqScaleTypes, Model::model_rep(), Minimizer::modify_n2s(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Iterator::numContinuousVars, Iterator::numContinuousConstraints, Minimizer::primaryRespScaleFlag, Minimizer::primaryRespScaleFlag, Minimizer::print_scaling(), Iterator::probDescDB, Model::response_labels(), Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, Minimizer::secondaryRespScaleFlag, RecastModel::submodel_supports_derivative_estimation(), Model::subordinate_model(), Model::supports_derivative_estimation(), and Minimizer::varsScaleFlag.

Referenced by Minimizer::scale_model().

13.56.3.18  void variables_scaler (const Variables & scaled_vars, Variables & native_vars) [static, private]

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(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Minimizer::minimizerInstance, Minimizer::modify_s2n(), Iterator::outputLevel, and Dakota::write_data().

Referenced by Minimizer::scale_model().

13.56.3.19  void secondary_resp_scaler (const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response) [static, private]

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 Minimizer::minimizerInstance, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, and Minimizer::response_scaler_core().

Referenced by Minimizer::scale_model().

13.56.3.20 RealVector modify_n2s (const RealVector & native_vars, const IntArray & scale_types, const RealVector & multipliers, const RealVector & offsets) const [private]

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 Minimizer::initialize_scaling().

13.56.3.21 void response_modify_n2s (const Variables & native_vars, const Response & native_response, Response & recast_response, int start_offset, int num_responses) const [private]

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::continuous_variable_ids(), Variables::continuous_variables(), Dakota::copy_data(), Variables::cv(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_labels(), Response::function_value(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Minimizer::numUserPrimaryFns, Iterator::outputLevel, Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, Dakota::write_col_vector_trans(), Dakota::write_data(), and Dakota::write_precision.

Referenced by Minimizer::response_scaler_core().

13.56.3.22 RealMatrix lin_coeffs_modify_n2s (const RealMatrix & src_coeffs, const RealVector & cv_multipliers, const RealVector & lin_multipliers) const [private]

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 Minimizer::initialize_scaling().

The documentation for this class was generated from the following files:

- DakotaMinimizer.hpp
- DakotaMinimizer.cpp
13.57 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*

**Protected Member Functions**

- **void reshape** (const SizetArray &vc_totals)
  
  *reshape the lower/upper bound arrays within the Constraints hierarchy*

- **void build_active_views** ()
  
  *construct active views of all variables bounds arrays*

- **void build_inactive_views** ()
  
  *construct inactive views of all variables bounds arrays*
13.57 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).

13.57.2 Constructor & Destructor Documentation

13.57.2.1 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, Constraints::build_views(), SharedVariablesData::components_totals(), Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Constraints::manage_linear_constraints(), Constraints::numLinearEqCons, Constraints::numLinearIneqCons, Constraints::sharedVarsData, SharedVariablesData::vc_lookup(), and SharedVariablesData::view().

13.57.3 Member Function Documentation

13.57.3.1 void reshape (const SizetArray & vc_totals) [protected, virtual]

reshape the lower/upper bound arrays within the Constraints hierarchy Resizes the derived bounds arrays.

Reimplemented from Constraints.

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, and Constraints::build_views().

Referenced by MixedVarConstraints::MixedVarConstraints().

The documentation for this class was generated from the following files:

- MixedVarConstraints.hpp
- MixedVarConstraints.cpp
13.58 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) const
  write a variables object to an std::ostream
- void **write_aprepro** (std::ostream &s) const
  write a variables object to an std::ostream in aprepro format
- void **read_tabular** (std::istream &s)
- void **write_tabular** (std::ostream &s) const
  write a variables object in tabular format to an std::ostream
- void **reshape** (const SizetArray &vc_totals)
  reshapes an existing Variables object based on the incoming variablesComponents
- void **build_active_views** ()
  construct active views of all variables arrays
- void **build_inactive_views** ()
construct inactive views of all variables arrays

13.58.1 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)).

13.58.2 Constructor & Destructor Documentation

13.58.2.1 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::build_views(), SharedVariablesData::components_totals(), Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Variables::sharedVarsData, and SharedVariablesData::vc_lookup().

13.58.3 Member Function Documentation

13.58.3.1 void read_tabular (std::istream & s) [protected, virtual]

Presumes variables object is already appropriately sized to receive!

Reimplemented from Variables.

References Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, and Dakota::read_data_tabular().

The documentation for this class was generated from the following files:

- MixedVariables.hpp
- MixedVariables.cpp
13.59 Model Class Reference

Base class for the model class hierarchy. Inheritance diagram for Model:

```
Model
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NestedModel</td>
</tr>
<tr>
<td>RecastModel</td>
</tr>
<tr>
<td>SingleModel</td>
</tr>
<tr>
<td>SurrogateModel</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
</tr>
<tr>
<td>HierarchSurrModel</td>
</tr>
</tbody>
</table>
```

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 ()**
  
  *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.*

- **virtual Model & surrogate_model ()**
  
  *return the approximation sub-model in surrogate models*

- **virtual Model & truth_model ()**
  
  *return the truth sub-model in 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 (bool recurse_flag=true)**
propagate vars/labels/bounds/targets from the bottom up

- virtual Interface & interface()
  return the interface employed by the derived model class, if present: SingleModel::userDefinedInterface, DataFitSurrModel::approxInterface, or NestedModel::optionalInterface

- 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 void build_approximation ()
  build a new SurrogateModel approximation

- virtual bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)
  build a new SurrogateModel approximation using/enforcing response at vars

- virtual void update_approximation (bool rebuild_flag)
  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)
  replace the anchor point data within an existing surrogate

- virtual void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  replace the data points within an existing surrogate

- virtual void append_approximation (bool rebuild_flag)
  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)
  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)
  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 restore_approximation()

- virtual void restore_approximation ()
  restore a previous approximation data state within a surrogate
• virtual bool restore_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 ()
  move the current approximation into storage for later combination

• virtual void combine_approximation (short corr_type)
  combine the current approximation with one previously stored

• virtual bool force_rebuild ()
  determine whether a surrogate model rebuild should be forced based on changes in the inactive data

• virtual std::vector< Approximation > & approximations ()
  retrieve the set of Approximations within a DataFitSurrModel

• virtual const Pecos::SurrogateData & approximation_data (size_t index)
  retrieve the approximation data from a particular Approximation instance within a DataFitSurrModel

• virtual const RealVectorArray & approximation_coefficients ()
  retrieve the approximation coefficients from each Approximation within a DataFitSurrModel

• virtual void approximation_coefficients (const RealVectorArray & approx_coeffs)
  set the approximation coefficients for each Approximation within a DataFitSurrModel

• virtual const RealVector & approximation_variances (const Variables & vars)
  retrieve the approximation 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 DiscrepancyCorrection & discrepancy_correction ()
  return the DiscrepancyCorrection object used by SurrogateModels

• virtual void component_parallel_mode (short mode)
  update component parallel mode for supporting parallelism in a model's interface component, sub-model component, or neither component [componentParallelMode = 0 (none), 1 (INTERFACE/APPROX_-INTERFACE/OPTIONAL_INTERFACE/LF_MODEL/SURROGATE_MODEL), or 2 (SUB_MODEL/ACTUAL_-MODEL/HF_MODEL/TRUTH_MODEL)].

• virtual String local_eval_synchronization ()
return derived model synchronization setting

• virtual int local_eval_concurrency ()
  return derived model asynchronous evaluation concurrency

• virtual void serve (int max_iterator_concurrency)
  Service job requests received from the master. Completes when a termination message is received from stop_servers().

• virtual void stop_servers ()
  Executed by the master to terminate all server operations for a particular model when iteration on the model is complete.

• virtual bool derived_master_overload () const
  Return a flag indicating the combination of multiprocessor evaluations and a dedicated master iterator scheduling. Used in synchronous compute_response functions to prevent the error of trying to run a multiprocessor job on the master.

• virtual void inactive_view (short view, bool recurse_flag=true)
  update the Model's inactive view based on higher level (nested) context

• virtual const String & interface_id () const
  return the interface identifier

• virtual int evaluation_id () const
  Return the value of the evaluation id counter for the Model.

• virtual bool evaluation_cache () const
  Indicates the usage of an evaluation cache by the Model.

• virtual void set_evaluation_reference ()
  Set the reference points for the evaluation counters within the Model.

• virtual void fine_grained_evaluation_counters ()
  Request fine-grained evaluation reporting within the Model.

• virtual void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  Print an evaluation summary for the Model.

• virtual void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix

• ModelList & subordinate_models (bool recurse_flag=true)
  return the sub-models in nested and surrogate models

• void compute_response ()
Compute the **Response** at currentVariables (default **ActiveSet**).

- **void compute_response** (const **ActiveSet** &set)
  
  Compute the **Response** at currentVariables (specified **ActiveSet**).

- **void asynch_compute_response** ()
  
  Spawn an asynchronous job (or jobs) that computes the value of the **Response** at currentVariables (default **ActiveSet**).

- **void asynch_compute_response** (const **ActiveSet** &set)
  
  Spawn an asynchronous job (or jobs) that computes the value of the **Response** at currentVariables (specified **ActiveSet**).

- **const IntResponseMap & synchronize** ()
  
  Execute a blocking scheduling algorithm to collect the complete set of results from a group of asynchronous evaluations.

- **const IntResponseMap & synchronize_nowait** ()
  
  Execute a nonblocking scheduling algorithm to collect all available results from a group of asynchronous evaluations.

- **void init_communicators** (int max_iterator_concurrency, bool recurse_flag=true)
  
  allocate communicator partitions for a model and store configuration in modelPCIterMap

- **void init_serial** ()
  
  for cases where **init_communicators**() will not be called, modify some default settings to behave properly in serial.

- **void set_communicators** (int max_iterator_concurrency, bool recurse_flag=true)
  
  set active parallel configuration for the model (set modelPCIter from modelPCIterMap)

- **void free_communicators** (int max_iterator_concurrency, bool recurse_flag=true)
  
  deallocate communicator partitions for a model

- **void stop_configurations** ()
  
  called from **Strategy::init_iterator**() for iteratorComm rank 0 to terminate **serve_configurations**() on other iteratorComm processors

- **int serve_configurations** ()
  
  called from **Strategy::init_iterator**() for iteratorComm rank != 0 to balance **init_communicators**() calls on iteratorComm rank 0

- **void estimate_message_lengths** ()
  
  estimate messageLengths for a model

- **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 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 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 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

• 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

• 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

- const RealVector & discrete_real_variables() const
  return the active discrete real variables 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

- UShortMultiArrayConstView discrete_int_variable_types() const
  return the active discrete variable types from currentVariables

- UShortMultiArrayConstView discrete_real_variable_types() const
  return the active discrete variable types from currentVariables

- SizetMultiArrayConstView continuous_variable_ids() const
  return the active continuous variable identifiers from 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

- 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
  return the inactive continuous variable identifiers from currentVariables`

```cpp
const RealVector & all_continuous_variables () const
  return all continuous variables in currentVariables
```

```cpp
void all_continuous_variables (const RealVector &a_c_vars)
  set all continuous variables in currentVariables
```

```cpp
void all_continuous_variable (Real a_c_var, size_t i)
  set a variable within the all continuous variables in currentVariables
```

```cpp
const IntVector & all_discrete_int_variables () const
  return all discrete variables in currentVariables
```

```cpp
void all_discrete_int_variables (const IntVector &a_d_vars)
  set all discrete variables in currentVariables
```

```cpp
void all_discrete_int_variable (int a_d_var, size_t i)
  set a variable within the all discrete variables in currentVariables
```

```cpp
const RealVector & all_discrete_real_variables () const
  return all discrete variables in currentVariables
```

```cpp
void all_discrete_real_variables (const RealVector &a_d_vars)
  set all discrete variables in currentVariables
```

```cpp
void all_discrete_real_variable (Real a_d_var, size_t i)
  set a variable within the all discrete variables in currentVariables
```

```cpp
UShortMultiArrayConstView all_continuous_variable_types () const
  return all continuous variable types from currentVariables
```

```cpp
UShortMultiArrayConstView all_discrete_int_variable_types () const
  return all discrete variable types from currentVariables
```

```cpp
UShortMultiArrayConstView all_discrete_real_variable_types () const
  return all discrete variable types from currentVariables
```

```cpp
SizetMultiArrayConstView all_continuous_variable_ids () const
  return all continuous variable identifiers from currentVariables
```

```cpp
const IntSetArray & discrete_design_set_int_values () const
  return the sets of values available for each of the discrete design set integer variables
```
• void discrete_design_set_int_values (const IntSetArray &isa)
  define the sets of values available for each of the discrete design set integer 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 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

• const IntSetArray & discrete_set_int_values ()
  return the sets of values available for each of the active discrete set integer variables (aggregated in activeDiscSetIntValues)

• const RealSetArray & discrete_set_real_values ()
  return the sets of values available for each of the active discrete set real variables (aggregated in activeDiscSetRealValues)

• 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_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_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_label (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_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

• 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

• 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

- 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

- 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

- 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

- 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
to return the inactive discrete lower bounds in userDefinedConstraints

• void inactive_discrete_int_lower_bounds(const IntVector &i_d_l_bnds)
to set the inactive discrete lower bounds in userDefinedConstraints

• const IntVector & inactive_discrete_int_upper_bounds() const
to return the inactive discrete upper bounds in userDefinedConstraints

• void inactive_discrete_int_upper_bounds(const IntVector &i_d_u_bnds)
to set the inactive discrete upper bounds in userDefinedConstraints

• const RealVector & inactive_discrete_real_lower_bounds() const
to return the inactive discrete lower bounds in userDefinedConstraints

• void inactive_discrete_real_lower_bounds(const RealVector &i_d_l_bnds)
to set the inactive discrete lower bounds in userDefinedConstraints

• const RealVector & inactive_discrete_real_upper_bounds() const
to return the inactive discrete upper bounds in userDefinedConstraints

• void inactive_discrete_real_upper_bounds(const RealVector &i_d_u_bnds)
to set the inactive discrete upper bounds in userDefinedConstraints

• const RealVector & all_continuous_lower_bounds() const
to return all continuous lower bounds in userDefinedConstraints

• void all_continuous_lower_bounds(const RealVector &a_c_l_bnds)
to set all continuous lower bounds in userDefinedConstraints

• void all_continuous_lower_bound(Real a_c_l_bnd, size_t i)
to set a lower bound within continuous lower bounds in userDefinedConstraints

• const RealVector & all_continuous_upper_bounds() const
to return all continuous upper bounds in userDefinedConstraints

• void all_continuous_upper_bounds(const RealVector &a_c_u_bnds)
to set all continuous upper bounds in userDefinedConstraints

• void all_continuous_upper_bound(Real a_c_u_bnd, size_t i)
to set an upper bound within all continuous upper bounds in userDefinedConstraints

• const IntVector & all_discrete_int_lower_bounds() const
to 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
• void linear_ineq_constraint_lower_bounds (const RealVector &lin_ineq_l_bnds)
  return 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

• const String & gradient_type () const
  return the gradient evaluation type (gradType)

• const String & method_source () const
  return the numerical gradient evaluation method source (methodSrc)

• 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 (fdGradSS)

• const String & fd_gradient_step_type () const
  return the finite difference gradient step type (fdGradST)

• 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 (hessType)

• 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 (fdHessByGradSS)

• const RealVector & fd_hessian_by_fn_step_size () const
  return function-based finite difference Hessian step size (fdHessByFnSS)

• const String & fd_hessian_step_type () const
  return the finite difference Hessian step type (fdHessST)

• 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

• bool derivative_estimation ()
indicates potential usage of estimate_derivatives() based on gradType/hessType

- void supports_derivative_estimation (bool sed_flag)
  set whether this model should perform or pass on derivative estimation

- void init_comms_bcast_flag (bool icb_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 (const ParConfigLIter &pc_iter)
  set modelPCIter

- const ParConfigLIter & parallel_configuration_iterator () const
  return modelPCIter

- void auto_graphics (const bool flag)
  set modelAutoGraphicsFlag to activate posting of graphics data within compute_response/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

- Real initialize_h (Real x_j, Real lb_j, Real ub_j, Real step_size, String step_type)
function to determine initial finite difference $h$ (before step length adjustment) based on type of step desired

- Real FDstep1 (Real $x_0_j$, Real $lb_j$, Real $ub_j$, Real $h_{mag}$)
  function returning finite-difference step size (affected by bounds)

- Real FDstep2 (Real $x_0_j$, Real $lb_j$, Real $ub_j$, Real $h$)
  function returning second central-difference step size (affected by bounds)

Public Attributes

- bool shortStep
  flags finite-difference step size adjusted by bounds

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 (NoDBBaseConstructor, ParallelLibrary &parallel_lib, const SharedVariablesData &svd, const ActiveSet &set, short output_level)
  constructor initializing base class for derived model class instances constructed on the fly

- Model (RecastBaseConstructor, ProblemDescDB &problem_db, ParallelLibrary &parallel_lib)
  constructor initializing base class for recast model class instances constructed on the fly

- virtual void derived_compute_response (const ActiveSet &set)
  portion of compute_response() specific to derived model classes

- virtual void derived_asynch_compute_response (const ActiveSet &set)
  portion of asynch_compute_response() 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

- virtual void derived_init_communicators (int max_iterator_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 (int max_iterator_concurrency, bool recurse_flag=true)  
portion of set_communicators() specific to derived model classes

virtual void derived_free_communicators (int max_iterator_concurrency, bool recurse_flag=true)  
portion of free_communicators() specific to derived model classes

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 modelType  
type of model: single, nested, or surrogate

String surrogateType  
type of surrogate model: local_*, multipoint_*, global_*, or hierarchical

String gradType  
grad type: none,numerical,analytic,mixed

String methodSrc  
method source: dakota,vendor

String intervalType  
interval type: forward,central

bool ignoreBounds  
option to ignore bounds when computing < finite differences
- `bool centralHess`
  option to use old 2nd-order finite diffs for Hessians

- `RealVector fdGradSS`
  relative step sizes for numerical gradients

- `String fdGradST`
  step type for numerical gradients

- `IntSet gradIdAnalytic`
  analytic id's for mixed gradients

- `IntSet gradIdNumerical`
  numerical id's for mixed gradients

- `String hessType`
  Hess type: none, numerical, quasi, analytic, mixed.

- `String quasiHessType`
  quasi-Hessian type: bfgs, damped_bfgs, sr1

- `RealVector fdHessByGradSS`
  relative step sizes for numerical Hessians < estimated with 1st-order grad differences

- `RealVector fdHessByFnSS`
  relative step sizes for numerical Hessians < estimated with 2nd-order fn differences

- `String fdHessST`
  step type for numerical Hessians

- `IntSet hessIdAnalytic`
  analytic id's for mixed Hessians

- `IntSet hessIdNumerical`
  numerical id's for mixed Hessians

- `IntSet hessIdQuasi`
  quasi id's for mixed Hessians

- `bool supportsEstimDerivs`
  whether model should perform or forward < derivative estimation

- `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/LF_MODEL), or 2 (SUB_MODEL/HF_-MODEL/TRUE_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

- **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

- **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

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_fn_hessians)
  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)
  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)
  return the upper bound for a finite difference offset, drawn from global or distribution bounds

• bool manage_asv (const ShortArray &asv_in, 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.
Private Attributes

- **String** `modelId`  
  model identifier string from the input file

- **int** `modelEvalCntr`  
  evaluation counter for top-level `compute_response()` and `asynch_compute_response()` 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 `asynch_compute_response()`

- **std::map<int, ParConfigIter>** `modelPCIterMap`  
  map<> used for tracking modelPCIter instances using concurrency level as the lookup key

- **bool** `initCommsBcastFlag`  
  flag for determining need to bcast the max concurrency from `init_communicators();` set from `Strategy::init_iterator()`

- **bool** `modelAutoGraphicsFlag`  
  flag for posting of graphics data within `compute_response` (automatic graphics posting in the model as opposed to graphics posting at the strategy level)

- **ModelList** `modelList`  
  used to collect sub-models for `subordinate_models()`

- **VariablesList** `varsList`  
  history of vars populated in `asynch_compute_response()` and used in `synchronize()`.

- **std::list<ShortArray>** `asvList`  
  if `estimate_derivatives()` is used, transfers ASVs from `asynch_compute_response()` to `synchronize()`

- **std::list<ActiveSet>** `setList`  
  if `estimate_derivatives()` is used, transfers ActiveSets from `asynch_compute_response()` 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`.

- **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** `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

- **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

- **Model** `modelRep`  
  pointer to the letter (initialized only for the envelope)

- **int** `referenceCount`  
  number of objects sharing `modelRep`
13.59 Model Class Reference

13.59.1 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 iterator 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.

13.59.2 Constructor & Destructor Documentation

13.59.2.1 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.

13.59.2.2 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.

13.59.2.3 Model (const Model & model)
copy constructor Copy constructor manages sharing of modelRep and incrementing of referenceCount.
References Model::modelRep, and Model::referenceCount.

13.59.2.4 ~Model () [virtual]
destructor Destructor decrements referenceCount and only deletes modelRep when referenceCount reaches zero.
References Model::modelRep, and Model::referenceCount.

13.59.2.5 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::fdGradSS, Model::fdHessByFnSS, Model::fdHessByGradSS, ProblemDescDB::get_sa(), Model::gradIdNumerical, Model::gradType,
Model::hessIdNumerical, Model::hessType, Response::num_functions(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Model::primaryRespFnSense, Dakota::strbegins(), Dakota::strtolower(), and Model::userDefinedConstraints.

13.59.2.6 Model (RecastBaseConstructor, ProblemDescDB & problem_db, ParallelLibrary & parallel_lib) [protected]

constructor initializing base class for recast model class instances constructed on the fly. 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. Note that parallel_lib is managed separately from problem_db since parallel_lib is needed even in cases where problem_db is an empty envelope (i.e., use of dummy_db in Model(NoDBBaseConstructor) above.

13.59.3 Member Function Documentation

13.59.3.1 Model operator=(const Model & model)


References Model::modelRep, and Model::referenceCount.

13.59.3.2 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 DataFitSurrModel, NestedModel, and RecastModel.

References Dakota::dummy_iterator, Model::modelRep, and Model::subordinate_iterator().

13.59.3.3 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 refer-
ence 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 NestedModel, RecastModel, and SurrogateModel.

References Dakota::dummy_model, Model::modelRep, and Model::subordinate_model().

Referenced by Minimizer::data_transform_model(), NonDGGlobalReliability::expected_feasibility(), NonDGGlobalReliability::expected_improvement(), SurrogateModel::force_rebuild(), NonDExpansion::initialize_expansion(), Minimizer::initialize_run(), Minimizer::initialize_scaling(), NonDExpansion::initialize_u_space_model(), NonDGGlobalReliability::optimize_gaussian_process(), LeastSq::post_run(), COLINOptimizer::post_run(), Optimizer::primary_resp_reducer(), LeastSq::primary_resp_weighter(), Optimizer::print_results(), LeastSq::print_results(), Minimizer::resize_best_resp_array(), Minimizer::resize_best_vars_array(), Minimizer::scale_model(), Model::subordinate_model(), DataFitSurrModel::update_global(), and LeastSq::weight_model().

13.59.3.4 Model & surrogate_model () [virtual]

return the 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 DataFitSurrModel, HierarchSurrModel, and RecastModel.

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(), SurrBasedLocalMinimizer::find_center_approx(), SurrBasedLocalMinimizer::minimize_surrogates(), SurrBasedGlobalMinimizer::minimize_surrogates(), NonDAdaptiveSampling::output_round_data(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), RecastModel::surrogate_model(), and Model::surrogate_model().

13.59.3.5 Model & truth_model () [virtual]

return the 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 DataFitSurrModel, HierarchSurrModel, and RecastModel.

References Dakota::dummy_model, Model::modelRep, and Model::truth_model().

Referenced by SurrogateModel::force_rebuild(), SurrBasedMinimizer::initialize_graphics(), SurrBasedLocalMinimizer::minimize_surrogates(), SurrBasedGlobalMinimizer::minimize_surrogates(), RecastModel::truth_model(), and Model::truth_model().

13.59.3.6 void update_from_subordinate_model (bool recurse_flag = true) [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). Single, Hierarchical, and Nested Models do not redefine the function since they do not support instantiate-on-the-fly. This means that the re-
cursion 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 DataFitSurrModel, and RecastModel.

References Model::modelRep, and Model::update_from_subordinate_model().

Referred by NonDLocalReliability::initialize_class_data(), NonDExpansion::initialize_expansion(),
Optimizer::initialize_run(), LeastSq::initialize_run(), EffGlobalMinimizer::minimize_surrogates_on_model(),
NonDGlobalReliability::optimize_gaussian_process(), NonDLocalInterval::quantify_uncertainty(),
NonDGlobalInterval::quantify_uncertainty(), RecastModel::update_from_subordinate_model(),
DataFitSurrModel::update_from_subordinate_model(), and Model::update_from_subordinate_model().

13.59.3.7 Interface & interface () [virtual]

return the interface employed by the derived model class, if present: SingleModel::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 DataFitSurrModel, NestedModel, RecastModel, and SingleModel.

References Dakota::dummy_interface, Model::interface(), and Model::modelRep.

Referred by RecastModel::interface(), Model::interface(), and SurrBasedGlobalMinimizer::minimize_surrogates().

13.59.3.8 String local_eval_synchronization () [virtual]

return derived model synchronization setting SingleModels 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, and SingleModel.

References Model::local_eval_synchronization(), and Model::modelRep.

Referred by Model::init_serial(), RecastModel::local_eval_synchronization(), Model::local_eval_synchronization(), and Model::set_communicators().

13.59.3.9 int local_eval_concurrency () [virtual]

return derived model asynchronous evaluation concurrency SingleModels and HierarchSurrModels redefine this virtual function.

Reimplemented in RecastModel, and SingleModel.

References Model::local_eval_concurrency(), and Model::modelRep.
13.59 Model Class Reference

Referenced by RecastModel::local_eval_concurrency(), Model::local_eval_concurrency(), and Model::set_communicators().

13.59.3.10  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 DataFitSurrModel, NestedModel, RecastModel, and SingleModel.
References Dakota::dummy_interface, Interface::interface_id(), Model::interface_id(), and Model::modelRep.
Referenced by DataFitSurrModel::build_global(), DataFitSurrModel::DataFitSurrModel(), Model::estimate_derivatives(), Model::estimate_message_lengths(), SurrBasedLocalMinimizer::find_center_approx(), RecastModel::interface_id(), Model::interface_id(), Optimizer::local_objective_recast_retrieve(), SNLLLeastSq::post_run(), SurrBasedMinimizer::print_results(), Optimizer::print_results(), LeastSq::print_results(), SequentialHybridStrategy::run_sequential(), DiscrepancyCorrection::search_db(), Analyzer::update_best(), SequentialHybridStrategy::update_local_results(), ConcurrentStrategy::update_local_results(), and NonDLocalReliability::update_mpp_search_data().

13.59.3.11  bool evaluation_cache () 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 SingleModel.
References Model::evaluation_cache(), and Model::modelRep.
Referenced by DataFitSurrModel::DataFitSurrModel(), and Model::evaluation_cache().

13.59.3.12  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 DataFitSurrModel, HierarchSurrModel, NestedModel, RecastModel, and SingleModel.
References Model::eval_tag_prefix(), and Model::modelRep.
Referenced by HierarchSurrModel::build_approximation(), HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), HierarchSurrModel::derived_compute_response(), DataFitSurrModel::derived_compute_response(), RecastModel::eval_tag_prefix(), Model::eval_tag_prefix(), and Iterator::eval_tag_prefix().

13.59.3.13  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 DataFitSurrModel::build_global(), NLPQLPOptimizer::initialize(), NCSUOptimizer::initialize(),
DOTOptimizer::initialize(), CONMINOptimizer::initialize(), NonDLocalInterval::NonDLocalInterval(), NonD-
LocalReliability::NonDLocalReliability(), SOLBase::SOLBase(), Model::subordinate_models(), and SurrBased-
LocalMinimizer::SurrBasedLocalMinimizer().

13.59.3.14 void init_communicators (int max_iterator_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 message-
Lengths 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_i(), 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(), and Model::parallelLib.

Referenced by APPSOptimizer::APPSOptimizer(), COLINOptimizer::COLINOptimizer(),
NonDExpansion::construct_expansion_sampler(), RecastModel::derived_init_communicators(),
NestedModel::derived_init_communicators(), HierarchSurrModel::derived_init_communicators(),
DataFitSurrModel::derived_init_communicators(), EffGlobalMinimizer::EffGlobalMinimizer(),
Model::init_communicators(), EfficientSubspaceMethod::init_fullspace_sampler(), Strategy::init_-
iterator(), JEGAOptimizer::JEGAOptimizer(), LeastSq::LeastSq(), NonDLocalReliability::method_-
recourse(), NonDLocalInterval::method_recourse(), NonDAdaptiveSampling::NonDAdaptiveSampling(),
NonDBayesCalibration::NonDBayesCalibration(), NonDGlobalInterval::NonDGlobalInterval(),
NonDGlobalReliability::NonDGlobalReliability(), NonDGPImpSampling::NonDGPImpSampling(),
NonDPolynomialChaos::NonDPolynomialChaos(), NonDStochCollocation::NonDStochCollocation(),
Optimizer::Optimizer(), EfficientSubspaceMethod::reduced_space_uq(), Model::serve_configurations(),
SNLLOptimizer::SNLLOptimizer(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), and SurrBasedLo-
calMinimizer::SurrBasedLocalMinimizer().

13.59.3.15 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 RecastModel::derived_init_serial(), NestedModel::derived_init_serial(),
HierarchSurrModel::derived_init_serial(), DataFitSurrModel::derived_init_serial(), and Model::init_serial().
13.59.3.16  void estimate_message_lengths ()

estimate_message_lengths 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 message_lengths 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(), Response::copy(), Model::currentResponse, Model::currentVariables, Model::estimate_message_lengths(), Model::interface_id(), Model::message_lengths, Model::modelRep, Model::numFns, Model::parallelLib, MPIPackBuffer::reset(), MPIPackBuffer::size(), and ParallelLibrary::world_size().

Referenced by ConcurrentStrategy::ConcurrentStrategy(), Model::estimate_message_lengths(), and Model::init_communicators().

13.59.3.17  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::modelRep, and Model::referenceCount.

Referenced by Minimizer::data_transform_model(), EffGlobalMinimizer::EffGlobalMinimizer(), NonDAdaptiveSampling::NonDAdaptiveSampling(), NonDBayesCalibration::NonDBayesCalibration(), NonDGlobalInterval::NonDGlobalInterval(), NonDGGlobalReliability::NonDGGlobalReliability(), NonDGPolynomialSampling::NonDGPolynomialSampling(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDPolynomialChaos::NonDPolynomialChaos(), NonDStochCollocation::NonDStochCollocation(), Optimizer::reduce_model(), EfficientSubspaceMethod::reduced_space_uq(), Minimizer::scale_model(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), NonD::transform_model(), and LeastSq::weight_model().

13.59.3.18  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::gradType, Model::hessIdNumerical, Model::hessType, Model::intervalType, Model::methodSrc, Model::modelRep, and Model::numDerivVars.
13.59.3.19 Real initialize_h (Real x_j, Real lb_j, Real ub_j, Real step_size, String step_type)

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().

13.59.3.20 Real FDstep1 (Real x0_j, Real lb_j, Real ub_j, Real h_mag)

function returning finite-difference step size (affected by bounds) Auxiliary function to compute forward or first
central-difference step size.
References Model::ignoreBounds, and Model::shortStep.
Referenced by Model::estimate_derivatives().

13.59.3.21 Real FDstep2 (Real x0_j, Real lb_j, Real ub_j, Real h)

function returning second central-difference step size (affected by bounds) Auxiliary function to second central-
difference step size, honoring bounds.
References Model::ignoreBounds, and Model::shortStep.
Referenced by Model::estimate_derivatives().

13.59.3.22 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().

13.59.3.23 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 differ-
ence 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.

! new logic

References Model::all_continuous_lower_bounds(), Model::all_continuous_upper_bounds(), Model::all_continuous_variable_ids(), Model::all_continuous_variable_types(), Variables::all_continuous_variables(), Model::centralHess, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variable_ids(), Variables::continuous_variable_ids(), Model::continuous_variable_types(), Variables::continuous_variables(), Response::copy(), Dakota::copy_data(), Model::currentResponse, Model::currentVariables, Dakota::data_pairs, Model::dbCaptureList, Model::dbResponseList, Model::deltaList, ActiveSet::derivative_vector(), Model::derived_asynch_compute_response(), Model::derived_compute_response(), Model::fdGradSS, Model::fdHessByFnSS, Model::fdHessByGradSS, Model::fdHessST, Model::FDstep1(), Model::FDstep2(), Dakota::find_index(), Model::finite_difference_lower_bound(), Model::finite_difference_upper_bound(), Response::function_gradients(), Response::function_values(), Model::ignoreBounds, Model::inactive_continuous_lower_bounds(), Model::inactive_continuous_upper_bounds(), Model::inactive_continuous_variable_ids(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variable_types(), Variables::inactive_continuous_variables(), Model::initialize_h(), Model::initialMapList, Model::interface_id(), Model::intervalType, Dakota::lookup_by_val(), Model::numFns, Model::outputLevel, ActiveSet::request_vector(), Model::shortStep, and Model::update_response().

Referenced by Model::asynch_compute_response(), and Model::compute_response().

13.59.3.24 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 compute_response() for the case of asynchronous estimate_derivatives() and by synchronize() for the case where one or more asynch_compute_response() calls has employed asynchronous estimate_derivatives().

!

References Response::active_set(), Model::acv(), Variables::all_continuous_variable_ids(), Model::centralHess, Variables::continuous_variable_ids(), Response::copy(), 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, ActiveSet::request_values(), Response::reset_inactive(), and Model::update_response().

Referenced by Model::compute_response(), and Model::synchronize().

13.59.3.25 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_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::hessIdQuasi, Model::hessType, 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().

13.59.3.26 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(), Dakota::copy_data(), Model::fnGradsPrev, Response::function_gradients(), Model::hessIdQuasi, Model::hessType, 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().

13.59.3.27 bool manage_asv (const ShortArray & asv_in, 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(), Model::gradIdAnalytic, Model::gradIdNumerical, Model::gradType, Model::hessIdAnalytic, Model::hessIdNumerical, Model::hessIdQuasi, Model::hessType, Model::intervalType, Model::methodSrc, Model::supportsEstimDerivs, and Model::surrogate_response_mode().

Referenced by Model::asynch_compute_response(), and Model::compute_response().

The documentation for this class was generated from the following files:

- DakotaModel.hpp
- DakotaModel.cpp
13.60 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`.

• void `pack` (const u_long &data)
  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.

13.60.1 Detailed Description

Class for packing MPI message buffers. A class that provides a facility for packing message buffers using the MPI_Pack 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
13.61 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.

• void unpack (bool *data, const int num=1)
  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.

13.61.1 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
13.62 NCSUOptimizer Class Reference

Wrapper class for the NCSU DIRECT optimization library. Inheritance diagram for NCSUOptimizer:

```
NCSUOptimizer
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
    |          |
verify problem respects NCSU DIRECT Fortran limits

Static Private Member Functions

- static int objective_eval (int *n, double c[], double l[], double u[], int point[], int *maxI, int *start, int *maxfunc, double fvec[], int iidata[], int *iisize, double ddata[], int *idsize, char cdata[], int *icsize)

  'fep' in Griffin-modified NCSUDirect: computes the value of the objective function (potentially at multiple points, passed by function pointer to NCSUDirect). Include unscaling from DIRECT.

Private Attributes

- short setUpType
  controls iteration mode: SETUP_MODEL (normal usage) or SETUP_USERFUNC (user-supplied functions mode for "on the fly" instantiations). see enum in NCSUOptimizer.cpp NonDGlobalsReliability currently uses the model mode. GaussProcApproximation currently uses the user_functions mode.

- Real minBoxSize
  holds the minimum boxsize

- Real volBoxSize
  hold the minimum volume boxsize

- Real solutionTarget
  holds the solution target minimum to drive towards

- RealVector lowerBounds
  holds variable lower bounds passed in for "user_functions" mode.

- RealVector upperBounds
  holds variable upper bounds passed in for "user_functions" mode.

- double(* userObjectiveEval )(const RealVector &x)
  holds function pointer for objective function evaluator passed in for "user_functions" mode.

Static Private Attributes

- static NCSUOptimizer * ncsudirectInstance
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
13.62.1 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:

13.62.2 Constructor & Destructor Documentation

13.62.2.1 NCSUOptimizer (Model & model)

standard constructor This is the standard constructor with method specification support.

References NCSUOptimizer::check_inputs(), and NCSUOptimizer::initialize().

13.62.2.2 NCSUOptimizer (Model & model, const int & max_iter, const int & max_eval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX)

alternate constructor for instantiations "on the fly" This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

References NCSUOptimizer::check_inputs(), NCSUOptimizer::initialize(), Iterator::maxFunctionEvals, and Iterator::maxIterations.

13.62.2.3 NCSUOptimizer (NoDBBaseConstructor, Model & model)

alternate constructor for Iterator instantiations by name This is an alternate constructor for Iterator instantiations by name using a Model but no ProblemDescDB.

References NCSUOptimizer::check_inputs(), and NCSUOptimizer::initialize().

13.62.2.4 NCSUOptimizer (const RealVector & var_l_bnds, const RealVector & var_u_bnds, const int & max_iter, const int & max_eval, double(*)(const RealVector &x) user_obj_eval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX)

alternate constructor for instantiations "on the fly" This is an alternate constructor for performing an optimization using the passed in objective function pointer.

References NCSUOptimizer::check_inputs(), Iterator::maxFunctionEvals, and Iterator::maxIterations.
13.62.3 Member Function Documentation

13.62.3.1 int objective_eval (int * n, double c[], double l[], double u[], int * point, int * maxI, int * start, int * maxfunc, double fvec[], int iidata[], int * iisize, double ddata[], int * idsize, char cdata[], int * icsize) [static, private]

'fep' in Griffin-modified NCSUDirect: computes the value of the objective function (potentially at multiple points, passed by function pointer to NCSUDirect). Include unscaling from DIRECT. Modified batch evaluator that accepts multiple points and returns corresponding vector of functions in fvec. Must be used with modified DIRECT src (DIRbatch.f).

References Model::asynch_compute_response(), Model::asynch_flag(), Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_value(), Iterator::iteratedModel, NCSUOptimizer::ncsudirectInstance, Model::primary_response_fn_sense(), NCSUOptimizer::setUpType, Model::synchronize(), and NCSUOptimizer::userObjectiveEval.

Referenced by NCSUOptimizer::find_optimum().

The documentation for this class was generated from the following files:

- NCSUOptimizer.hpp
- NCSUOptimizer.cpp
13.63 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_compute_response (const ActiveSet &set)**
  
  *portion of compute_response() specific to NestedModel*

- **void derived_asynch_compute_response (const ActiveSet &set)**
  
  *portion of asynch_compute_response() 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 & interface ()**
  
  *return optionalInterface*

- **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*
• bool derived_master_overload () const
  flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)

• void derived_init_communicators (int max_iterator_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 (int max_iterator_concurrency, bool recurse_flag=true)
  set active parallel configuration within subModel

• void derived_free_communicators (int max_iterator_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the NestedModel (forwarded to optionalInterface and subModel)

• void serve (int max_iterator_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 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 subModel)

• 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)

• virtual void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix
Private Member Functions

- 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

- 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_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)

  offset cv_index to create index into aggregated primary/secondary arrays

- size_t **div_index_map** (size_t div_index)

  offset div_index to create index into aggregated primary/secondary arrays

- size_t **drv_index_map** (size_t drv_index)

  offset drv_index to create index into aggregated primary/secondary arrays

- size_t **ccv_index_map** (size_t ccv_index)

  offset active complement ccv_index to create index into all continuous arrays

- size_t **cdiv_index_map** (size_t cdiv_index)

  offset active complement cdiv_index to create index into all discrete int arrays

- size_t **cdrv_index_map** (size_t cdrv_index)

  offset active complement cdrv_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 **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

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 ()

update subModel with current variable values/bounds/labels

Private Attributes

int nestedModelEvalCntr

number of calls to derived_compute_response() / derived_async_compute_response()

Iterator subIterator

the sub-iterator that is executed on every evaluation of this model

Model subModel

the sub-model used in sub-iterator evaluations

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

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 insertions 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 insertions 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 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 complement1ADRVarMapIndices
  “primary” variable mappings for inserting the complement of the active discrete real currentVariables within all discrete real subModel variables.
• **BoolDeque** `extraCVarsData`
  flags for updating subModel continuous bounds and labels, one for each active continuous variable in `currentVariables`

• **BoolDeque** `extraDIVarsData`
  flags for updating subModel discrete int bounds and labels, one for each active discrete int variable in `currentVariables`

• **BoolDeque** `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.

• **String** `evalTagPrefix`
  cached evalTag Prefix from parents to use at compute_response time

### 13.63.1 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.

### 13.63.2 Member Function Documentation

#### 13.63.2.1 `void derived_compute_response (const ActiveSet & set)` [protected, virtual]

portion of `compute_response()` 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 **Response**: `active_set()`, **NestedModel**: `component_parallel_mode()`, **Model**: `currentResponse`, **Model**: `currentVariables`, **Iterator**: `eval_tag_prefix()`, **Interface**: `eval_tag_prefix()`, **NestedModel**: `evalTagPrefix`, **Model**: `hierarchicalTagging`, **Interface**: `map()`, **NestedModel**: `nestedModelEvalCntr`, **NestedModel**: `optInterfaceResponse`, **NestedModel**: `optionalInterface`, **NestedModel**: `response_mapping()`, **Iterator**: `response_results()`, **Iterator**: `response_results_active_set()`, **Iterator**: `run_iterator()`, **NestedModel**: `set_mapping()`, **NestedModel**: `subIterator`, and **NestedModel**: `update_sub_model()`.

---

Dakota Version 5.4 Developers Manual generated on November 7, 2013
13.63.2.2  void derived_asynch_compute_response (const ActiveSet & set)  [protected, virtual]  
portion of asynch_compute_response() specific to NestedModel. Not currently supported by NestedModels (need to add concurrent iterator support). As a result, derived_synchronize() and derived_synchronize_nowait() are inactive as well.
Reimplemented from Model.
References Dakota::abort_handler(), Response::active_set(), Model::currentResponse, Model::currentVariables, Interface::map(), NestedModel::nestedModelEvalCntr, NestedModel::optInterfaceResponse, NestedModel::optionalInterface, and NestedModel::set_mapping().

13.63.2.3  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.compute_response() within subIterator.run().
Reimplemented from Model.
References Interface::iterator_eval_dedicated_master_flag(), Interface::multi_proc_eval_flag(), NestedModel::optInterfacePointer, and NestedModel::optionalInterface.

13.63.2.4  void derived_init_communicators (int max_iterator_concurrency, bool recurse_flag = true)  
[inline, protected, virtual]  
set up optionalInterface and subModel for parallel operations. Asynchronous flags need to be initialized for the subModel. In addition, max_iterator_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 Model::init_communicators(), Interface::init_communicators(), Iterator::maximum_concurrency(), Model::messageLengths, NestedModel::optInterfacePointer, NestedModel::optionalInterface, NestedModel::subIterator, and NestedModel::subModel.

13.63.2.5  int 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.

13.63.2.6  size_t cv_index_map (size_t cv_index)  [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-real variables.
References Model::currentVariables, Variables::variables_components_totals(), and Variables::view().

Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

13.63.2.7 size_t div_index_map (size_t div_index) [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-real variables.

References Model::currentVariables, Variables::cv(), Variables::variables_components_totals(), and Variables::view().

Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

13.63.2.8 size_t drv_index_map (size_t drv_index) [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-real variables.

References Model::currentVariables, Variables::cv(), Variables::div(), Variables::variables_components_totals(), and Variables::view().

Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

13.63.2.9 size_t ccv_index_map (size_t ccv_index) [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(), Model::currentVariables, Variables::variables_components_totals(), and Variables::view().

Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

13.63.2.10 size_t cdiv_index_map (size_t cdiv_index) [private]

offset active complement cdiv_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(), Model::currentVariables, Variables::variables_components_totals(), and Variables::view().

Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

13.63.2.11 size_t cdv_index_map (size_t cdv_index) [private]

offset active complement cdv_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(), Model::currentVariables, Variables::variables_components_totals(), and Variables::view().
**13.63 NestedModel Class Reference**

Referenced by NestedModel::NestedModel(), and NestedModel::update_sub_model().

### 13.63.2.12 void response_mapping (const Response & opt_interface_response, const Response & sub_iterator_response, Response & mapped_response) [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,

```plaintext
optionalInterface fns = {f}, {g} (deterministic primary functions, constraints)
subIterator fns = (S) (UQ response statistics)
```

Problem formulation for mapped functions:

```plaintext
minimize \( f \) + \( W \)\{S\}
subject to \( \{g_l\} \leq \{g\} \leq \{g_u\} \)
\( \{a_l\} \leq \{A\}\{S\} \leq \{a_u\} \)
\( \{g\} = \{g_t\} \)
\( \{A\}\{S\} = \{a_t\} \)
```

where \( [W] \) is the primary_mapping_matrix user input (primaryRespCoeffs class attribute), \( [A] \) is the secondary_mapping_matrix user input (secondaryRespCoeffs class attribute), \( \{\{g_l\},\{a_l\}\} \) 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() [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: \( \{g_l\} \leq \{g\} + \{A\}\{S\} \leq \{g_u\} \) 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 Dakota::abort_handler(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), Dakota::copy_data(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessians(), Response::function_values(), Response::function_values_view(), Response::num_functions(), NestedModel::numOptInterfEqCon, NestedModel::numOptInterfIneqCon, NestedModel::numOptInterfPrimary, NestedModel::numSubIterFns, NestedModel::numSubIterMappedEqCon, NestedModel::numSubIterMappedIneqCon, NestedModel::optInterfacePointer, NestedModel::primaryRespCoeffs, Response::reset_inactive(), and NestedModel::secondaryRespCoeffs.

Referenced by NestedModel::derived_compute_response().

---

Dakota Version 5.4 Developers Manual generated on November 7, 2013
13.63.3 Member Data Documentation

13.63.3.1 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_free_communicators(), NestedModel::derived_init_communicators(), NestedModel::derived_init_serial(), NestedModel::derived_set_communicators(), NestedModel::derived_subordinate_models(), 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::serve(), NestedModel::set_mapping(), NestedModel::sm_acv_index_map(), NestedModel::sm_adiv_index_map(), NestedModel::subordinate_model(), NestedModel::surrogate_response_mode(), NestedModel::update_inactive_view(), and NestedModel::update_sub_model().

The documentation for this class was generated from the following files:

- NestedModel.hpp
- NestedModel.cpp
13.64  NIDRProblemDescDB Class Reference

The derived input file database utilizing the new IDR parser. Inheritance diagram for NIDRProblemDescDB::

```
    ProblemDescDB
    |   
    |    NIDRProblemDescDB
```

Public Member Functions

- **NIDRProblemDescDB** (ParallelLibrary &parallel_lib)
  constructor

- **~NIDRProblemDescDB** ()
  destructor

- void **derived_parse_inputs** (const char *dakota_input_file, const char *parser_options)
  parses the input file and populates the problem description database using NIDR.

- void **derived_broadcast** ()
  perform any data processing that must be coordinated with DB buffer broadcasting (performed prior to broadcasting the DB buffer on rank 0 and after receiving the DB buffer on other processor ranks)

- void **derived_post_process** ()
  perform any additional data post-processing

- **KWH** (iface_Rlit)
- **KWH** (iface_false)
- **KWH** (iface_ilit)
- **KWH** (iface_pint)
- **KWH** (iface_lit)
- **KWH** (iface_start)
- **KWH** (iface_stop)
- **KWH** (iface_str)
- **KWH** (iface_str2D)
- **KWH** (iface_strL)
- **KWH** (iface_true)
- **KWH** (method_li)
- **KWH** (method_Real)
- **KWH** (method_Real01)
- **KWH** (method_RealDL)
- **KWH** (method_RealLlit)
- **KWH** (method_Realp)
• `KWH (method_Realz)`
• `KWH (method_Ri)`
• `KWH (method_false)`
• `KWH (method_szarray)`
• `KWH (method_lil2)`
• `KWH (method_lil2p)`
• `KWH (method_int)`
• `KWH (method_ivec)`
• `KWH (method_lit)`
• `KWH (method_lit2)`
• `KWH (method_litc)`
• `KWH (method_liti)`
• `KWH (method_litp)`
• `KWH (method_litr)`
• `KWH (method_litz)`
• `KWH (method_nnint)`
• `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_uharray)`
• `KWH (method_ushint)`
• `KWH (method_type)`
• `KWH (model_Real)`
• `KWH (model_RealDL)`
• `KWH (model_false)`
• `KWH (model_int)`
• `KWH (model_intsetm1)`
• `KWH (model_lit)`
• `KWH (model_order)`
• `KWH (model_shint)`
• `KWH (model_start)`
• `KWH (model_stop)`
• `KWH (model_str)`
• `KWH (model_strL)`
• KWH (model_true)
• KWH (model_type)
• KWH (resp_RealDL)
• KWH (resp_RealL)
• KWH (resp_false)
• KWH (resp_intset)
• KWH (resp_ivec)
• KWH (resp_lit)
• KWH (resp_sizet)
• KWH (resp_start)
• KWH (resp_stop)
• KWH (resp_str)
• KWH (resp_strL)
• KWH (resp_true)
• KWH (strategy_Real)
• KWH (strategy_RealL)
• KWH (strategy_int)
• KWH (strategy_lit)
• KWH (strategy_start)
• KWH (strategy_str)
• KWH (strategy_strL)
• KWH (strategy_true)
• KWH (var_RealLb)
• KWH (var_RealUb)
• KWH (var_caulbl)
• KWH (var_dauilbl)
• KWH (var_daurlbl)
• KWH (var_ceulbl)
• KWH (var_deuilbl)
• 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_newivec)
• KWH (var_newrvec)
• KWH (var_ivec)
• KWH (var_rvec)
• KWH (var_type)
Static Public Member Functions

- static void `botch` (const char *fmt,...)
- static void `check_variables` (std::list< DataVariables > *)
- static void `check_responses` (std::list< DataResponses > *)
- static void `make_variable_defaults` (std::list< DataVariables > *)
- static void `make_response_defaults` (std::list< DataResponses > *)
- static void `squawk` (const char *fmt,...)
- static void `warn` (const char *fmt,...)

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
- static int nerr = 0

Static Private Member Functions

- static void `check_variables_node` (void *)

Private Attributes

- std::list< void * > VIL

13.64.1 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://www.sandia.gov/~dmgay/nidr08.pdf). Source for the routines declared herein is NIDRProblemDescDB.cpp, in which most routines are so short that a description seems unnecessary.

13.64.2 Member Function Documentation

13.64.2.1 void derived_parse_inputs (const char *dakota_input_file, const char *parser_options) [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(), ProblemDescDB::dataInterfaceList, ProblemDescDB::dataMethodList, DataMethodRep::dlDetails, DataMethodRep::dlLib, ProblemDescDB::parallel_library(), and NIDRProblemDescDB::pDDBInstance.

The documentation for this class was generated from the following files:

- NIDRProblemDescDB.hpp
- NIDRProblemDescDB.cpp
13.65 NL2Res Struct Reference

Auxiliary information passed to calcr and calcj via ur.

Public Attributes

- Real * r
  
  residual \( r = r(x) \)

- Real * J
  
  Jacobian \( J = J(x) \).

- Real * x
  
  corresponding parameter vector

- int nf
  
  function invocation count for \( r(x) \)

13.65.1 Detailed Description

Auxiliary information passed to calcr and calcj via ur.

The documentation for this struct was generated from the following file:

- NL2SOLLeastSq.cpp
13.66 NL2SOLLeastSq Class Reference

Wrapper class for the NL2SOL nonlinear least squares library. Inheritance diagram for NL2SOLLeastSq:

```
  Iterator
    Minimizer
      LeastSq
        NL2SOLLeastSq
```

Public Member Functions

- `NL2SOLLeastSq (Model &model)`
  standard constructor

- `NL2SOLLeastSq (NoDBBaseConstructor, Model &model)`
  alternate constructor

- `~NL2SOLLeastSq ()`
  destructor

- `void minimize_residuals ()`
  Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function for the least squares branch.

Static Private Member Functions

- `static void calcr (int *np, int *pp, Real *x, int *nfp, Real *r, int *ui, void *ur, Vf vf)`
  evaluator function for residual vector

- `static void calcj (int *np, int *pp, Real *x, int *nfp, Real *J, int *ui, void *ur, Vf vf)`
  evaluator function for residual Jacobian

Private Attributes

- `int auxprt`
  auxiliary printing bits (see Dakota Ref Manual): sum of $< 1 = x0prt$ (print initial guess) $< 2 = solprt$ (print final solution) $< 4 = statpr$ (print solution statistics) $< 8 = parprt$ (print nondefault parameters) $< 16 = dradpr$ (print bound constraint drops/adds) $<$ debug/verbose/normal use default $= 31$ (everything), $<$ quiet uses 3, silent uses 0.
- int `outlev`
  frequency of output summary lines in number of iterations < (debug/verbose/normal/quiet use default = 1, silent uses 0)

- Real `dltfdj`
  finite-diff step size for computing Jacobian approximation < (fd_gradient_step_size)

- Real `delta0`
  finite-diff step size for gradient differences for H < (a component of some covariance approximations, if desired) < (fd_hessian_step_size)

- Real `dltdfc`
  finite-diff step size for function differences for H < (fd_hessian_step_size)

- int `mxfcal`
  function-evaluation limit (max_function_evaluations)

- int `mxiter`
  iteration limit (max_iterations)

- Real `rfctol`
  relative fn convergence tolerance (convergence_tolerance)

- Real `afctol`
  absolute fn convergence tolerance (absolute_conv_tol)

- Real `xctl`
  x-convergence tolerance (x_conv_tol)

- Real `sctol`
  singular convergence tolerance (singular_conv_tol)

- Real `lmaxs`
  radius for singular-convergence test (singular_radius)

- Real `xftol`
  false-convergence tolerance (false_conv_tol)

- int `covreq`
  kind of covariance required (covariance): < 1 or -1 ==> sigma^2 H^-1 J^T J H^-1 < 2 or -2 ==> sigma^2 H^-1 < 3 or -3 ==> sigma^2 (J^T J)^-1 < 1 or 2 == use function diffs to estimate H < -1 or -2 == use gradient diffs to estimate H < default = 0 (no covariance)

- int `rdreq`
  whether to compute the regression diagnostic vector < (regression_diagnostics)
13.66 NL2SOLLeastSq Class Reference

- Real fprec
  
  _expected response function precision_ (function_precision)

- Real lmax0
  
  _initial trust-region radius_ (initial_trust_radius)

Static Private Attributes

- static NL2SOLLeastSq * nl2solInstance
  
  _pointer to the active object instance used within the static evaluator functions_

13.66.1 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 dnfgb from Bell Labs; see [http://www.netlib.org/port/readme](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.

13.66.2 Member Function Documentation

13.66.2.1 void minimize_residuals () [virtual]

Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function for the least squares branch.


Implements LeastSq.

References NL2SOLLeastSq::afctol, NL2SOLLeastSq::auxprt, Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::boundConstraintFlag, NL2SOLLeastSq::calcj(), NL2SOLLeastSq::calcr(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Dakota::copy_data(), NL2SOLLeastSq::covreq, NL2SOLLeastSq::delta0, NL2SOLLeastSq::dltfdc, NL2SOLLeastSq::dltfdj, NL2SOLLeastSq::fprec, LeastSq::get_confidence_intervals(), Iterator::gradientType, Iterator::iteratedModel, NL2SOLLeastSq::lmax0, NL2SOLLeastSq::lmaxs, NL2SOLLeastSq::mxfcal, NL2SOLLeastSq::mxiter, NL2SOLLeastSq::nl2solInstance, Iterator::numContinuousVars, LeastSq::numLeastSqTerms, NL2SOLLeastSq::outlev, NL2SOLLeastSq::rdreq, NL2SOLLeastSq::rfctol, NL2SOLLeastSq::scxtol, Minimizer::speculativeFlag, Minimizer::vendorNumericalGradFlag, NL2SOLLeastSq::xctol, and NL2SOLLeastSq::xtol.

The documentation for this class was generated from the following files:

- NL2SOLLeastSq.hpp
- NL2SOLLeastSq.cpp
13.67 NLPQLPOptimizer Class Reference

Wrapper class for the NLPQLP optimization library, Version 2.0. Inheritance diagram for NLPQLPOptimizer::

```
        Iterator
          |     |
          v     v
Minimizer
          |     |
          v     v
Optimizer
          |     |
          v     v
NLPQLPOptimizer
```

Public Member Functions

- **NLPQLPOptimizer (Model &model)**
  - *standard constructor*

- **NLPQLPOptimizer (NoDBBaseConstructor, Model &model)**
  - *alternate constructor*

- **~NLPQLPOptimizer ()**
  - *destructor*

- **void find_optimum ()**
  - *Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- **void initialize_run ()**
  - *performs run-time set up*

Private Member Functions

- **void initialize ()**
  - *Shared constructor code.*

- **void allocate_workspace ()**
  - *Allocates workspace for the optimizer.*
• void deallocate_workspace ()
  Releases workspace memory.

• void allocate_constraints ()
  Allocates constraint mappings.

Private Attributes

• int L
  L: Number of parallel systems, i.e. function calls during line search at predetermined iterates. HINT: If only less than 10 parallel function evaluations are possible, it is recommended to apply the serial version by setting L=1.

• int numEqConstraints
  numEqConstraints: Number of equality constraints.

• int MMAX
  MMAX: Row dimension of array DG containing Jacobian of constraints. MMAX must be at least one and greater or equal to M.

• int N
  N: Number of optimization variables.

• int NMAX
  NMAX: Row dimension of C. NMAX must be at least two and greater than N.

• int MNN2
  MNN2: Must be equal to M+N+N+2.

• double * X
  X(NMAX,L): Initially, the first column of X has to contain starting values for the optimal solution. On return, X is replaced by the current iterate. In the driving program the row dimension of X has to be equal to NMAX. X is used internally to store L different arguments for which function values should be computed simultaneously.

• double * F
  F(L): On return, F(1) contains the final objective function value. F is used also to store L different objective function values to be computed from L iterates stored in X.

• double * G
  G(MMAX,L): On return, the first column of G contains the constraint function values at the final iterate X. In the driving program the row dimension of G has to be equal to MMAX. G is used internally to store L different set of constraint function values to be computed from L iterates stored in X.

• double * DF
  DF(NMAX): DF contains the current gradient of the objective function. In case of numerical differentiation and a distributed system (L>1), it is recommended to apply parallel evaluations of F to compute DF.
• double * DG

DG(MMAX,NMAX) : DG contains the gradients of the active constraints (ACTIVE(J)=.true.) at a current iterate X. The remaining rows are filled with previously computed gradients. In the driving program the row dimension of DG has to be equal to MMAX.

• double * U

U(MNN2) : U contains the multipliers with respect to the actual iterate stored in the first column of X. The first M locations contain the multipliers of the M nonlinear constraints, the subsequent N locations the multipliers of the lower bounds, and the final N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.

• double * C

C(NMAX,NMAX) : On return, C contains the last computed approximation of the Hessian matrix of the Lagrangian function stored in form of an LDL decomposition. C contains the lower triangular factor of an LDL factorization of the final quasi-Newton matrix (without diagonal elements, which are always one). In the driving program, the row dimension of C has to be equal to NMAX.

• double * D

D(NMAX) : The elements of the diagonal matrix of the LDL decomposition of the quasi-Newton matrix are stored in the one-dimensional array D.

• double ACC

ACC : The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be smaller than the accuracy by which gradients are computed.

• double ACCQP

ACCQP : The tolerance is needed for the QP solver to perform several tests, for example whether optimality conditions are satisfied or whether a number is considered as zero or not. If ACCQP is less or equal to zero, then the machine precision is computed by NLPQLP and subsequently multiplied by 1.0D+4.

• double STPMIN

STPMIN : Minimum steplength in case of L>1. Recommended is any value in the order of the accuracy by which functions are computed. The value is needed to compute a steplength reduction factor by STPMIN**(1/L-1). If STPMIN<0, then STPMIN=ACC is used.

• int MAXFUN

MAXFUN : The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20). MAXFUN is only needed in case of L=1, and must not be greater than 50.

• int MAXIT

MAXIT : Maximum number of outer iterations, where one iteration corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g. 100).

• int MAX_NM

MAX_NM : Stack size for storing merit function values at previous iterations for non-monotone line search (e.g. 10). In case of MAX_NM=0, monotone line search is performed.

• double TOL_NM
TOL_NM : Relative bound for increase of merit function value, if line search is not successful during the very first step. Must be non-negative (e.g. 0.1).

- `int IPRINT`

  IPRINT : Specification of the desired output level. IPRINT = 0 : No output of the program. IPRINT = 1 : Only a final convergence analysis is given. IPRINT = 2 : One line of intermediate results is printed in each iteration. IPRINT = 3 : More detailed information is printed in each iteration step, e.g. variable, constraint and multiplier values. IPRINT = 4 : In addition to 'IPRINT=3', merit function and steplength values are displayed during the line search.

- `int MODE`

  MODE : The parameter specifies the desired version of NLPQLP. MODE = 0 : Normal execution (reverse communication!). MODE = 1 : The user wants to provide an initial guess for the multipliers in U and for the Hessian of the Lagrangian function in C and D in form of an LDL decomposition.

- `int IOUT`

  IOUT : Integer indicating the desired output unit number, i.e. all write-statements start with 'WRITE(IOUT,... '.

- `int IFAIL`

  IFAIL : The parameter shows the reason for terminating a solution process. Initially IFAIL must be set to zero. On return IFAIL could contain the following values: IFAIL = -2 : Compute gradient values w.r.t. the variables stored in first column of X, and store them in DF and DG. Only derivatives for active constraints ACTIVE(J)=.TRUE. need to be computed. Then call NLPQLP again, see below. IFAIL = -1 : Compute objective fn and all constraint values subject the variables found in the first L columns of X, and store them in F and G. Then call NLPQLP again, see below. IFAIL = 0 : The optimality conditions are satisfied. IFAIL = 1 : The algorithm has been stopped after MAXIT iterations. IFAIL = 2 : The algorithm computed an uphill search direction. IFAIL = 3 : Underflow occurred when determining a new approximation matrix for the Hessian of the Lagrangian. IFAIL = 4 : The line search could not be terminated successfully. IFAIL = 5 : Length of a working array is too short. More detailed error information is obtained with 'IPRINT>0'. IFAIL = 6 : There are false dimensions, for example M>MMAX, N>=NMAX, or MNN2<M+N+N+2. IFAIL = 7 : The search direction is close to zero, but the current iterate is still infeasible. IFAIL = 8 : The starting point violates a lower or upper bound. IFAIL = 9 : Wrong input parameter, i.e., MODE, LDL decomposition in D and C (in case of MODE=1). IPRINT, IOUT IFAIL = 10 : Internal inconsistency of the quadratic subproblem, division by zero. IFAIL > 100 : The solution of the quadratic programming subproblem has been terminated with an error message and IFAIL is set to IFQL+100, where IFQL denotes the index of an inconsistent constraint.

- `double * WA`

  WA(LWA) : WA is a real working array of length LWA.

- `int LWA`

  LWA : LWA value extracted from NLPQLP20.f.

- `int * KWA`

  KWA(LKWA) : The user has to provide working space for an integer array.

- `int LKWA`

  LKWA : LKWA should be at least N+10.

- `int * ACTIVE`
ACTIVE(LACTIV) : The logical array shows a user the constraints, which NLPQLP considers to be active at the last computed iterate, i.e. \( G(J,X) \) is active, if and only if \( \text{ACTIVE}(J)=\text{TRUE.}, \ J=1,...,M \).

- int LACTIVE

\( LACTIV \) : The length \( LACTIV \) of the logical array should be at least \( 2*M+10 \).

- int LQL

\( LQL \) : If \( LQL = \text{TRUE.} \), the quadratic programming subproblem is to be solved with a full positive definite quasi-Newton matrix. Otherwise, a Cholesky decomposition is performed and updated, so that the subproblem matrix contains only an upper triangular factor.

- int numNlpqlConstr

total number of constraints seen by NLPQL

- SizetList nonlinIneqConMappingIndices

a list of indices for referencing the DAKOTA nonlinear inequality constraints used in computing the corresponding NLPQL constraints.

- RealList nonlinIneqConMappingMultipliers

a list of multipliers for mapping the DAKOTA nonlinear inequality constraints to the corresponding NLPQL constraints.

- RealList nonlinIneqConMappingOffsets

a list of offsets for mapping the DAKOTA nonlinear inequality constraints to the corresponding NLPQL constraints.

- SizetList linIneqConMappingIndices

a list of indices for referencing the DAKOTA linear inequality constraints used in computing the corresponding NLPQL constraints.

- RealList linIneqConMappingMultipliers

a list of multipliers for mapping the DAKOTA linear inequality constraints to the corresponding NLPQL constraints.

- RealList linIneqConMappingOffsets

a list of offsets for mapping the DAKOTA linear inequality constraints to the corresponding NLPQL constraints.

13.67.1 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

\[
\text{minimize } F(X) \text{ subject to } G(J,X) = 0, \ J=1,...,ME \ G(J,X) \geq 0, \ J=ME+1,...,M \ X_L \leq X \leq X_U
\]

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.

The documentation for this class was generated from the following files:

- NLPQLPOptimizer.hpp
- NLPQLPOptimizer.cpp
13.68 NLSSOLLeastSq Class Reference

Wrapper class for the NLSSOL nonlinear least squares library. Inheritance diagram for NLSSOLLeastSq:

![Inheritance Diagram for NLSSOLLeastSq](image)

**Public Member Functions**

- **NLSSOLLeastSq (Model &model)**
  
  *standard constructor*

- **NLSSOLLeastSq (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor*

- **~NLSSOLLeastSq ()**
  
  *destructor*

- **void minimize_residuals ()**
  
  *Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run virtual function for the least squares branch.***

**Static Private Member Functions**

- **static void least_sq_eval (int &mode, int &m, int &n, int &nrowfj, double *x, double *f, double *gradf, int &nstate)**

  *Evaluator for NLSSOL: computes the values and first derivatives of the least squares terms (passed by function pointer to NLSSOL).*

**Static Private Attributes**

- **static NLSSOLLeastSq * nlssolInstance**

  *pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data*
13.68 NLSSOLLeastSq Class Reference

13.68.1 Detailed Description

Wrapper class for the NLSSOL nonlinear least squares library. The NLSSOLLeastSq 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 NLSSOLLeastSq’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.

13.68.2 Constructor & Destructor Documentation

13.68.2.1 NLSSOLLeastSq (Model & model)

standard constructor This is the primary constructor. It accepts a Model reference.

References Minimizer::constraintTol, Iterator::convergenceTol, Iterator::fdGradStepSize, ProblemDescDB::get int(), ProblemDescDB::get real(), Iterator::gradientType, Iterator::maxIterations, Iterator::outputLevel, Iterator::probDescDB, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

13.68.2.2 NLSSOLLeastSq (NoDBBaseConstructor, Model & model)

alternate constructor This is an alternate constructor which accepts a Model but does not have a supporting method specification from the ProblemDescDB.

References Minimizer::constraintTol, Iterator::convergenceTol, Iterator::fdGradStepSize, Iterator::gradientType, Iterator::maxIterations, Iterator::outputLevel, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

The documentation for this class was generated from the following files:

- NLSSOLLeastSq.hpp
- NLSSOLLeastSq.cpp
13.69 NoDBBaseConstructor Struct Reference

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

Public Member Functions

- **NoDBBaseConstructor** (int=0)
  
  *C++ structs can have constructors.*

13.69.1 Detailed Description

Dummy struct for overloading constructors used in on-the-fly instantiations. **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
13.70 NomadOptimizer Class Reference

Wrapper class for NOMAD Optimizer. Inheritance diagram for NomadOptimizer::

Classes

- class Evaluator
  
  NOMAD-based Evaluator class.

Public Member Functions

- NomadOptimizer (Model &model)
  
  Constructor.

- NomadOptimizer (NoDBaseConstructor, Model &model)
  
  alternate constructor for Iterator instantiations by name

- ~NomadOptimizer ()
  
  Destructor.

- void find_optimum ()
  
  Calls the NOMAD solver.

Private Member Functions

- void load_parameters (Model &model)
  
  Convenience function for Parameter loading.
Private Attributes

- int `numTotalVars`
- int `numNomadNonlinearIneqConstraints`
- int `randomSeed`
- int `maxBlackBoxEvals`
- int `maxIterations`

  maximum number of iterations for the iterator

- std::string `outputFormat`
- std::string `historyFile`
- bool `displayAll`
- Real `epsilon`
- Real `vns`
- NOMAD::Point `initialPoint`
- NOMAD::Point `upperBound`
- NOMAD::Point `lowerBound`
- std::vector<int> `constraintMapIndices`

  map from Dakota constraint number to Nomad constraint number

- std::vector<double> `constraintMapMultipliers`

  multipliers for constraint transformations

- std::vector<double> `constraintMapOffsets`

  offsets for constraint transformations

13.70.1 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.
13.70.2 Constructor & Destructor Documentation

13.70.2.1 NomadOptimizer (Model & model)

Constructor. NOMAD Optimizer Constructor

Parameters:

  *model*  
  DAKOTA Model object

References ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), ProblemDescDB::get_string(), NomadOptimizer::load_parameters(), NomadOptimizer::maxIterations, and Iterator::probDescDB.

13.70.3 Member Function Documentation

13.70.3.1 void load_parameters (Model & model)  [private]

Convenience function for Parameter loading. This function takes the Parameters provided by the user in the DAKOTA model.

Parameters:

  *model*  
  NOMAD Model object

References Dakota::_NPOS, Dakota::abort_handler(), Minimizer::bigIntBoundSize, Minimizer::bigRealBoundSize, NomadOptimizer::constraintMapIndices, NomadOptimizer::constraintMapMultipliers, NomadOptimizer::constraintMapOffsets, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Model::discrete_int_lower_bounds(), Model::discrete_int_upper_bounds(), Model::discrete_int_variables(), Model::discrete_real_lower_bounds(), Model::discrete_real_upper_bounds(), Model::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Iterator::iteratedModel, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Iterator::numContinuousVars, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, and Dakota::set_value_to_index().

Referenced by NomadOptimizer::NomadOptimizer().

The documentation for this class was generated from the following files:

- NomadOptimizer.hpp
- NomadOptimizer.cpp
13.71 NonD Class Reference

Base class for all nondeterministic iterators (the DAKOTA/UQ branch). Inheritance diagram for NonD:

```
  Iterator
  |    |
  |    |
Analyzer
|      |
| NonD
```

```
  EfficientSubspaceMethod
  |    |
  |    |
NonDCalibration
|      |
| NonDExpansion
|      |
| NonDIntegration
|      |
| NonDInterval
|      |
| NonDPOFDarts
|      |
| NonDReliability
|      |
| 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)
  
  `alternate form: initialize natafTransform based on incoming data`

- 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_lev_tgt, short resp_lev_tgt_reduce, bool cdf_flag)
  
  `set requestedRespLevels, requestedProbLevels, requestedRelLevels, requestedGenRelLevels, respLevelTarget, and cdfFlag (used in combination with alternate ctors)`

- void `distribution_parameter_derivatives` (bool dist_param_derivs)
  
  `set distParamDerivs`
• bool pdf_output () const
  get pdfOutput

• void pdf_output (bool output)
  set pdfOutput

• Pecos::ProbabilityTransformation & variable_transformation ()
  return natafTransform

Protected Member Functions

• NonD (Model &model)
  constructor

• NonD (NoDBBaseConstructor, Model &model)
  alternate constructor for sample generation and evaluation "on the fly"

• NonD (NoDBBaseConstructor, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation "on the fly"

• ~NonD ()
  destructor

• void initialize_run ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

• void run ()
  run portion of run_iterator; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

• void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

• const Response & response_results () const
  return the final statistics from the nondeterministic iteration

• void response_results_active_set (const ActiveSet &set)
  set the active set within finalStatistics

• virtual void quantify_uncertainty ()=0
  performs a forward uncertainty propagation of parameter distributions into response statistics

• virtual void initialize_response_covariance ()
initializes respCovariance

• virtual void initialize_final_statistics ()
  initializes finalStatistics for storing NonD final results

• virtual void update_final_statistics ()
  update finalStatistics::functionValues

• int generate_system_seed ()
  create a system-generated unique seed (when a seed is unspecified)

• void initialize_random_variable_transformation ()
  instantiate natafTransform

• 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 ()
  verify that correlation warping supported by Der Kiureghian & Liu for given variable types

• void initialize_final_statistics_gradients ()
  initializes finalStatistics::functionGradients

• void update_aleatory_final_statistics ()
  update finalStatistics::functionValues from momentStats and computed{Prob,Rel,GenRel,Resp}Levels

• void update_system_final_statistics ()
  update system metrics from component metrics within finalStatistics

• void update_system_final_statistics_gradients ()
  update finalStatistics::functionGradients

• void initialize_distribution_mappings ()
  size computed{Resp,Prob,Rel,GenRel}Levels

• void print_distribution_mappings (std::ostream &s) const
  prints the z/p/beta/beta* mappings reflected in [requested, computed]{Resp,Prob,Rel,GenRel}Levels

• void print_system_mappings (std::ostream &s) const
print system series/parallel mappings for response levels

- void transform_model (Model &x_model, Model &u_model, bool global_bounds=false, Real bound=10.)
  recast x_model from x-space to u-space to create u_model

- void construct_lhs (Iterator &u_space_sampler, Model &u_model, const String &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 archive_allocate_mappings ()
  allocate results array storage for distribution mappings

- void archive_from_resp (size_t fn_index)
  archive the mappings from specified response levels for specified fn

- void archive_to_resp (size_t fn_index)
  archive the mappings to computed response levels for specified fn

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.

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 $X \rightarrow Z \rightarrow U$ 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 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 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 numHistogramPtVars
  number of histogram point 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 numDiscSetRealUncVars
  number of discrete real set uncertain variables (native space)
- `size_t numContAleatUncVars`
  total number of aleatory uncertain variables (native space)

- `size_t numDiscIntAleatUncVars`
  total number of aleatory uncertain variables (native space)

- `size_t numDiscRealAleatUncVars`
  total number of aleatory uncertain variables (native space)

- `size_t numAleatoryUncVars`
  total number of aleatory uncertain variables (native space)

- `size_t numContEpistUncVars`
  total number of epistemic uncertain variables (native space)

- `size_t numDiscIntEpistUncVars`
  total number of epistemic uncertain variables (native space)

- `size_t numDiscRealEpistUncVars`
  total number of 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`
  moments of response functions (mean, std deviation, skewness, and kurtosis calculated in `compute_moments()`), 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)

- Response finalStatistics
  final statistics from the uncertainty propagation used in strategies: response means, standard deviations, and probabilities of failure

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 `distribution_mappings_file` (size_t fn_index) const
  Write distribution mappings to a file for a single response.

- void `print_distribution_map` (size_t fn_index, std::ostream &s) const
  Print distribution mapping for a single response function to ostream.

**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.

**13.71.1 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.

**13.71.2 Member Function Documentation**

**13.71.2.1 void initialize_random_variables (short u_space_type)**
initialize 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(), NonD::natafTransform, and NonD::verify_correlation_support().

Referenced by NonDExpansion::compute_statistics(), NonDGlobalReliability::importance_sampling(), NonDLocalReliability::initialize_class_data(), NonDExpansion::initialize_expansion(), NonDAdaptImpSampling::NonDAdaptImpSampling(), and NonDGlobalReliability::optimize_gaussian_process().

**13.71.2.2 void initialize_random_variables (const Pecos::ProbabilityTransformation & transform)**
alternate form: initialize natafTransform based on incoming data This function is commonly used to publish tranformation 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, and NonD::numContStateVars.
13.71.2.3  **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 `Iterator`.

References `NonD::nondInstance`, and `NonD::prevNondInstance`.

13.71.2.4  **void run ()**  [inline, protected, virtual]

run portion of `run_iterator`; 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::bestVarsRespMap`, and `NonD::quantify_uncertainty()`.

13.71.2.5  **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 `NonD::nondInstance`, and `NonD::prevNondInstance`.

13.71.2.6  **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 means, standard deviations, and level mappings.

Reimplemented in `NonDInterval`.

References `Dakota::abort_handler()`, `NonD::cdfFlag`, `Model::cv()`, `ActiveSet::derivative_vector()`, `NonD::epistemicStats`, `NonD::finalStatistics`, `Response::function_labels()`, `Model::inactive_continuous_variable_ids()`, `Iterator::iteratedModel`, `Iterator::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()`, and `NonD::requested_levels()`.
13.71.2.7 **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.

References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Model::cv(), Iterator::iteratedModel, NonD::natafTransform, NonD::numBetaVars, NonD::numContDesVars, NonD::numContIntervalVars, NonD::numContStateVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNormalVars, NonD::numTriangularVars, NonD::numUniformVars, and NonD::numWeibullVars.

Referenced by NonDExpansion::initialize(), NonD::initialize_random_variables(), NonDBayesCalibration::NonDBayesCalibration(), NonDIntegration::NonDIntegration(), and NonDReliability::NonDReliability().

13.71.2.8 **void initialize_random_variable_parameters ()** [protected]

initializes ranVarMeansX, ranVarStdDevsX, ranVarLowerBndsX, ranVarUpperBndsX, and ranVarAddtParamsX 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::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::cv(), Iterator::iteratedModel, NonD::natafTransform, NonD::numBetaVars, NonD::numContDesVars, NonD::numContIntervalVars, NonD::numContStateVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNormalVars, NonD::numTriangularVars, NonD::numUniformVars, and NonD::numWeibullVars.

Referenced by NonDExpansion::initialize_expansion(), NonD::initialize_random_variables(), NonDLocalReliability::quantify_uncertainty(), NonDGlobalReliability::quantify_uncertainty(), and NonDBayesCalibration::quantify_uncertainty().

13.71.2.9 **void print_distribution_mappings (std::ostream & s) const** [protected]

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::distribution_mappings_file(), Iterator::numFunctions, Iterator::outputLevel, NonD::print_distribution_map(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, and Dakota::write_precision.

Referenced by NonDPOFDarts::print_results(), NonDGPImpSampling::print_results(), NonDExpansion::print_results(), NonDAdaptiveSampling::print_results(), NonDAdaptImpSampling::print_results(), and NonDSampling::print_statistics().
13.71.2.10 void vars_u_to_x_mapping (const Variables & u_vars, Variables & x_vars) [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().

13.71.2.11 void vars_x_to_u_mapping (const Variables & x_vars, Variables & u_vars) [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().

13.71.2.12 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().

13.71.2.13 void print_distribution_map (size_t fn_index, std::ostream & s) const [private]

Print distribution mapping for a single response function to ostream. Print the distribution mapping for a single response function to the passed output stream.

References NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRespLevels, Iterator::iteratedModel, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRespLevels, Model::response_labels(), and Dakota::write_precision.

Referenced by NonD::distribution_mappings_file(), and NonD::print_distribution_mappings().

The documentation for this class was generated from the following files:

- DakotaNonD.hpp
- DakotaNonD.cpp
13.72 NonDAdaptImpSampling Class Reference

Class for the Adaptive Importance Sampling methods within DAKOTA. Inheritance diagram for NonDAdaptImpSampling::

- Iterator
- Analyzer
- NonD
- NonDSampling
- NonDAdaptImpSampling

Public Member Functions

- NonDAdaptImpSampling (Model &model)
  
  **constructors**

- NonDAdaptImpSampling (Model &model, const String &sample_type, int samples, int seed, const String &rng, bool vary_pattern, short is_type, bool cdf_flag, bool x_space_data, bool x_space_model, bool bounded_model)

- ~NonDAdaptImpSampling ()
  
  **destructor**

- void quantify_uncertainty ()
  
  performs an adaptive importance sampling and returns probability of failure.

- void initialize (const RealVectorArray &initial_points, int resp_fn, const Real &initial_prob, const 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 &initial_point, int resp_fn, const Real &initial_prob, const 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

- const Real & get_probability ()
  
  returns the probability calculated by the importance sampling

- void print_results (std::ostream &s)
print the final statistics

Private Member Functions

- void converge_cov()
  iteratively generate samples and select representative points until coefficient of variation converges

- void converge_probability()
  iteratively generate samples from final set of representative points until probability converges

- void select_init_rep_points(const RealVectorArray &samples)
  select representative points from initial set of samples

- void select_rep_points(const RealVectorArray &samples)
  select representative points from a set of samples

- void calculate_rep_weights()
  calculate relative weights of representative points

- void generate_samples(RealVectorArray &samples)
  generate a set of samples based on multimodal sampling density

- void calculate_statistics(const RealVectorArray &samples, const size_t &total_sample_number, Real &probability_sum, Real &probability, bool cov_flag, Real &variance_sum, Real &coeff_of_variation)
  calculate the probability of exceeding the failure threshold and the coefficient of variation (if requested)

Private Attributes

- short importanceSamplingType
  integration type (is, ais, mmais) provided by input specification

- bool invertProb
  flag for inversion of probability values using 1.-p

- size_t numRepPoints
  the number of representative points around which to sample

- size_t respFn
  the response function in the model to be sampled

- RealVectorArray initPoints
  the original set of samples passed into the MMAIS routine
• RealVectorArray repPoints
  the set of representative points around which to sample

• RealVector repWeights
  the weight associated with each representative point

• RealVector designPoint
  design point at which uncertain space is being sampled

• bool transInitPoints
  flag to control if x->u transformation should be performed for initial points

• bool transPoints
  flag to control if u->x transformation should be performed before evaluation

• bool useModelBounds
  flag to control if the sampler should respect the model bounds

• bool initLHS
  flag to identify if initial points are generated from an LHS sample

• Real initProb
  the initial probability (from FORM or SORM)

• Real finalProb
  the final calculated probability (p)

• Real failThresh
  the failure threshold (z-bar) for the problem.

13.72.1 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.

13.72.2 Constructor & Destructor Documentation

13.72.2.1 NonDAdaptImpSampling (Model & model)

constructors standard constructor

This is the primary constructor. It accepts a Model reference.

References  ProblemDescDB::get_string(), NonDAdaptImpSampling::importanceSamplingType, NonD::initialize_random_variables(), Iterator::probDescDB, and NonDSampling::StatsFlag.
13.72.2.2 NonDAdaptImpSampling (Model & model, const String & sample_type, int samples, int seed, const String & rng, bool vary_pattern, short is_type, bool cdf_flag, bool x_space_data, bool x_space_model, bool bounded_model)

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB. References NonD::cdfFlag.

13.72.3 Member Function Documentation

13.72.3.1 void initialize (const RealVectorArray & initial_points, int resp_fn, const Real & initial_prob, const 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 set of starting points. References NonDAdaptImpSampling::designPoint, NonDAdaptImpSampling::failThres, NonDAdaptImpSampling::initPoints, NonDAdaptImpSampling::initProb, NonD::natafTransform, NonD::numContDesVars, Iterator::numContinuousVars, NonD::numUncertainVars, NonDAdaptImpSampling::respFn, and NonDAdaptImpSampling::transInitPoints.

Referenced by NonDExpansion::compute_statistics(), NonDGlobalReliability::importance_sampling(), and NonDAdaptImpSampling::quantify_uncertainty().

13.72.3.2 void initialize (const RealVector & initial_point, int resp_fn, const Real & initial_prob, const 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 Initializes data using only one starting point. References NonDAdaptImpSampling::designPoint, NonDAdaptImpSampling::failThres, NonDAdaptImpSampling::initPoints, NonDAdaptImpSampling::initProb, NonD::natafTransform, NonD::numContDesVars, Iterator::numContinuousVars, NonD::numUncertainVars, NonDAdaptImpSampling::respFn, and NonDAdaptImpSampling::transInitPoints.

The documentation for this class was generated from the following files:

- NonDAdaptImpSampling.hpp
- NonDAdaptImpSampling.cpp
13.73 NonDAdaptiveSampling Class Reference

Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate. Inheritance diagram for NonDAdaptiveSampling::

```
  Iterator
   Analy
      NonD
         NonDSampling
            NonDAdaptiveSampling
```

Public Member Functions

- **NonDAdaptiveSampling (Model &model)**
  
  *standard constructor*

- **~NonDAdaptiveSampling ()**
  
  *alternate constructor for sample generation and evaluation "on the fly" has not been implemented*

- **void print_results (std::ostream &s)**
  
  *print the final statistics*

Protected Member Functions

- **void quantify_uncertainty ()**
  
  *perform the GP importance sampling and return probability of failure.*

- **const Real & get_probability ()**
  
  *returns the probability calculated by the importance sampling*

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 computed 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_amsc (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, const String &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)
  compute 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.*

- **String sampleDesign**
  
  *String describing the initial sample design is based on. Options are: sampling_lhs, 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.*
13.73 NonDAdaptiveSampling Class Reference

13.73.1 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.

13.73.2 Constructor & Destructor Documentation

13.73.2.1 NonDAdaptiveSampling (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 NonDAdaptiveSampling::AMSC, NonDAdaptiveSampling::approx_type, Model::assign_rep(), Iterator::assign_rep(), NonDAdaptiveSampling::batchSize, NonDAdaptiveSampling::batchStrategy, NonDAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_sa(), ProblemDescDB::get_string(), NonDAdaptiveSampling::gpBuild, NonDAdaptiveSampling::gpEval, NonDAdaptiveSampling::gpFinalEval, NonDAdaptiveSampling::gpModel, Iterator::gradientType, Iterator::hessianType, Model::init_communicators(), Iterator::iteratedModel, Iterator::maximum_concurrency(), 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, NonDSampling::rngName, NonDAdaptiveSampling::sampleDesign, NonDAdaptiveSampling::scoringMetric, NonDSampling::vary_pattern(), and NonDSampling::varyPattern.

13.73.2.2 ~NonDAdaptiveSampling ()

alternate constructor for sample generation and evaluation "on the fly" has not been implemented destructor

References Model::free_communicators(), NonDAdaptiveSampling::gpEval, NonDAdaptiveSampling::gpModel, and Iterator::maximum_concurrency().

The documentation for this class was generated from the following files:

- NonDAdaptiveSampling.hpp
- NonDAdaptiveSampling.cpp
13.74 NonDBayesCalibration Class Reference

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

```
NonDBayesCalibration
  NonDCalibration
    NonD
      Analyzer
      Iterator
        NonDDREAMBayesCalibration
        NonDGPM SabhaBayesCalibration
        NonDQUESOBayesCalibration
```

Public Member Functions

- `NonDBayesCalibration(Model &model)`  
  `standard constructor`

- `~NonDBayesCalibration()`  
  `destructor`

Protected Member Functions

- `void quantify_uncertainty()`  
  `performs a forward uncertainty propagation of parameter distributions into response statistics`

- `const Model & algorithm_space_model()`  
  `return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain`

Protected Attributes

- `Model emulatorModel`  
  `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).`
13.74 NonDBayesCalibration Class Reference

- **bool standardizedSpace**
  
  *flag indicating use of a variable transformation to standardized probability space*

- **Iterator stochExpIterator**
  
  *NonDPolynomialChaos or NonDStochCollocation instance for defining a PCE/SC-based emulatorModel.*

- **Iterator lhsIterator**
  
  *LHS iterator for generating samples for GP.*

**Private Attributes**

- **short emulatorType**
  
  *the emulator type: NO_EMULATOR, GAUSSIAN_PROCESS, POLYNOMIAL_CHAOS, or STOCHASTIC_COLOCATION*

13.74.1 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.

13.74.2 Constructor & Destructor Documentation

13.74.2.1 NonDBayesCalibration (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 Iterator::algorithm_space_model(), Model::assign_rep(), Iterator::assign_rep(), NonD::cdfFlag, NonDBayesCalibration::emulatorModel, NonDBayesCalibration::emulatorType, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_ua(), Iterator::gradientType, Iterator::hessianType, Model::init_communicators(), NonD::initialize_random_variable_correlations(), NonD::initialize_random_variable_transformation(), NonD::initialize_random_variable_types(), Iterator::iteratedModel, Iterator::iterator_rep(), NonDBayesCalibration::lhsIterator, Iterator::outputLevel, Iterator::probDescDB, NonD::requested_levels(), NonD::respLevelTarget, NonD::respLevelTargetReduce, NonDBayesCalibration::standardizedSpace, NonDBayesCalibration::stochExpIterator, NonD::transform_model(), and NonD::verify_correlation_support().

The documentation for this class was generated from the following files:

- NonDBayesCalibration.hpp
- NonDBayesCalibration.cpp
13.75 NonDCalibration Class Reference

Inheritance diagram for NonDCalibration::

```
NonDAnalyzer
  NonD
    NonDCalibration
    NonDBayesCalibration
    NonDDREAMBayesCalibration
    NonDGPMSABayesCalibration
    NonDQUESOBayesCalibration

Iterator
```

Public Member Functions

- **NonDCalibration (Model &model)**
  
  *standard constructor*

- **~NonDCalibration ()**
  
  *destructor*

Protected Member Functions

- **void set_configuration_vars (Model &model, const RealVector &x)**
  
  *set the passed configuration variables into the model’s state vars*

Protected Attributes

- **RealVector expStdDeviations**
  
  *1 or numFunctions standard deviations*

- **String expDataFileName**
  
  *filename from which to read experimental data; optionally configuration vars x and standard deviations sigma*

- **bool expDataFileAnnotated**
  
  *whether the data file is in annotated format*
size_t numExperiments
    number of experiments to read from data file

IntVector numReplicates
    number of replicates per experiment

size_t numExpConfigVars
    number of columns in data file which are state variables

size_t numExpStdDeviationsRead
    how many sigmas to read from the data file (1 or numFunctions)

ExperimentData expData
    Container for experimental data to which to calibrate model.

Private Member Functions

bool find_state_index (unsigned short state_type, UShortMultiArrayConstView variable_types, std::string context_message, size_t &start_index)
    helper function to lookup a state_type enum variable type in the array of variables_types to find its start_index into the all array

Private Attributes

size_t continuousConfigVars
    number of continuous configuration variables

size_t discreteIntConfigVars
    number of discrete integer configuration variables

size_t discreteRealConfigVars
    number of discrete real configuration variables

size_t continuousConfigStart
    index of configuration variables in all continuous array

size_t discreteIntConfigStart
    index of configuration variables in all discrete integer array

size_t discreteRealConfigStart
    index of configuration variables in all discrete real array
13.75.1 Detailed Description

This class ...

13.75.2 Constructor & Destructor Documentation

13.75.2.1 NonDCalibration (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(), Model::all_continuous_variable_types(), Model::all_discrete_int_variable_types(), Model::all_discrete_real_variable_types(), NonDCalibration::continuousConfigStart, NonDCalibration::continuousConfigVars, NonDCalibration::discreteIntConfigStart, NonDCalibration::discreteIntConfigVars, NonDCalibration::discreteRealConfigStart, NonDCalibration::discreteRealConfigVars, NonDCalibration::expDataFileName, NonDCalibration::expStdDeviations, NonDCalibration::find_state_index(), ProblemDescDB::get_sizet(), Iterator::iteratedModel, NonDCalibration::numExpConfigVars, NonDCalibration::numExperiments, Iterator::numFunctions, NonDCalibration::numReplicates, and Iterator::probDescDB.

The documentation for this class was generated from the following files:

- NonDCalibration.hpp
- NonDCalibration.cpp
13.76 NonDCubature Class Reference

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

```
NonDCubature
\downarrow
NonDIntegration
\downarrow
NonD
\downarrow
Analyzer
\downarrow
Iterator
```

Public Member Functions

- `NonDCubature (Model &model, const Pecos::ShortArray &u_types, unsigned short cub_int_order)`
- `unsigned short integrand_order () const`
  
  
  ```c++
  return cubIntOrder
  ```

Protected Member Functions

- `NonDCubature (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)`
  
  ```
  Returns one block of samples (ndim * num_samples).
  ```

- `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`
get the current number of samples

Private Member Functions

- void \texttt{check\_integration} (const \texttt{Pecos::ShortArray} \&\texttt{u\_types}, const \texttt{Pecos::AleatoryDistParams} \&\texttt{adp})
  verify self-consistency of integration specification

- void \texttt{increment\_reference} ()
  increment each \texttt{cubIntOrderRef} entry by 1

Private Attributes

- \texttt{Pecos::CubatureDriver} \* \texttt{cubDriver}
  convenience pointer to the \texttt{numIntDriver} representation

- unsigned short \texttt{cubIntOrderRef}
  reference point for \texttt{Pecos::CubatureDriver::cubIntOrder}: the original user specification for the number of Gauss points per dimension, plus any refinements posted by \texttt{increment\_grid()}

- unsigned short \texttt{cubIntRule}
  the isotropic cubature integration rule

13.76.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals. This class is used by \texttt{NonDPolynomialChaos}, but could also be used for general numerical integration of moments. It employs Stroud cubature rules and extensions by D. Xiu.

13.76.2 Constructor & Destructor Documentation

13.76.2.1 \texttt{NonDCubature} (\texttt{Model} \& \texttt{model}, const \texttt{Pecos::ShortArray} \& \texttt{u\_types}, unsigned short \texttt{cub\_int\_order})

This alternate constructor is used for on-the-fly generation and evaluation of numerical cubature points.

References \texttt{Model\::aleatory\_distribution\_parameters()}, \texttt{NonDCubature\::check\_integration()}, \texttt{NonDCubature\::cubDriver}, \texttt{NonDCubature\::cubIntOrderRef}, \texttt{Iterator\::iteratedModel}, and \texttt{NonDIntegration\::numIntDriver}.

13.76.2.2 \texttt{NonDCubature} (\texttt{Model} \& \texttt{model}) [protected]

constructor This constructor is called for a standard letter-envelope iterator instantiation. In this case, \texttt{set\_db\_list\_nodes} has been called and \texttt{probDescDB} can be queried for settings from the method specification. It is not currently used, as there is not yet a separate \texttt{nond\_cubature} method specification.
13.76.3 Member Function Documentation

13.76.3.1 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.

13.76.3.2 void increment_grid_preference (const RealVector & dim_pref) [inline, protected, virtual]

Should not be used, but pure virtual must be defined.
Reimplemented from NonDIntegration.
References NonDCubature::increment_grid().

13.76.3.3 int num_samples () const [inline, protected, virtual]

get the current number of samples 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 maxConcurrency. May be (is) overridden by derived classes.
Reimplemented from Iterator.
References NonDCubature::cubDriver.

13.76.3.4 void increment_reference () [inline, private]

increment each cubIntOrderRef entry by 1 cubIntOrderRef 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
13.77  NonDDREAMBayesCalibration Class Reference

Bayesian inference using the DREAM approach. Inheritance diagram for NonDDREAMBayesCalibration::

![Inheritance diagram]

Public Member Functions

- **NonDDREAMBayesCalibration (Model &model)**
  *standard constructor*

- **~NonDDREAMBayesCalibration ()**
  *destructor*

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 DREAM*

- **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 DREAM.*

- **static double prior_density (int par_num, double zp[])**
  *Compute the prior density at specified point zp.*

- **static double * prior_sample (int par_num)**
  *Sample the prior and return an array of parameter values.*

- **static double sample_likelihood (int par_num, double zp[])**

Dakota Version 5.4 Developers Manual generated on November 7, 2013
 Likelihood function for call-back from DREAM to DAKOTA for evaluation.

Protected Member Functions

- void quantify_uncertainty ()
  redefined from DakotaNonD

Protected Attributes

- Real likelihoodScale
  scale factor for proposal covariance

- int numSamples
  number of samples in the chain (e.g. number of MCMC samples)

- bool calibrateSigmaFlag
  flag to indicate if the sigma terms should be calibrated (default true)

- int randomSeed
  random seed to pass to QUESO

- RealVector paramMins
  lower bounds on calibrated parameters

- RealVector paramMaxs
  upper bounds on calibrated parameters

- int numChains
  number of concurrent chains

- int numGenerations
  number of generations

- 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

- std::vector< boost::math::uniform > priorDistributions
  uniform prior PDFs for each variable

- boost::mt19937 rnumGenerator
  random number engine for sampling the prior

- std::vector< boost::uniform_real< double > > priorSamplers
  samplers for the uniform prior PDFs for each variable

Private Attributes

- short emulatorType
  the emulator type: NO_EMULATOR, GAUSSIAN_PROCESS, POLYNOMIAL_CHAOS, or STOCHASTIC_COLLOCATION

Static Private Attributes

- static NonDDREAMBayesCalibration * NonDDREAMInstance
  Pointer to current class instance for use in static callback functions.

13.77.1 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)

13.77.2 Constructor & Destructor Documentation

13.77.2.1 NonDDREAMBayesCalibration (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 NonDDREAMBayesCalibration::crossoverChainPairs, NonDDREAMBayesCalibration::grThreshold, NonDDREAMBayesCalibration::jumpStep, NonDDREAMBayesCalibration::numChains, NonDDREAMBayesCalibration::numCR, NonDDREAMBayesCalibration::numGenerations, and NonDDREAMBayesCalibration::numSamples.
**13.77.3 Member Function Documentation**

**13.77.3.1 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::NonDDREAMInstance, NonDDREAMBayesCalibration::numChains, Iterator::numContinuousVars, NonDDREAMBayesCalibration::numCR, and NonDDREAMBayesCalibration::numGenerations.

**13.77.3.2 double prior_density (int par_num, double zp[]) [static]**

Compute the prior density at specified point zp. See documentation in DREAM examples)

References NonDDREAMBayesCalibration::NonDDREAMInstance, and NonDDREAMBayesCalibration::priorDistributions.

**13.77.3.3 double * prior_sample (int par_num) [static]**

Sample the prior and return an array of parameter values. See documentation in DREAM examples)

References NonDDREAMBayesCalibration::NonDDREAMInstance, NonDDREAMBayesCalibration::priorSamplers, and NonDDREAMBayesCalibration::rnumGenerator.

**13.77.3.4 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 NonDDREAMBayesCalibration::calibrateSigmaFlag, Model::compute_response(), Model::continuous_variables(), Model::current_response(), NonDBayesCalibration::emulatorModel, NonDDREAMBayesCalibration::emulatorType, NonDCalibration::expData, Response::function_values(), NonDDREAMBayesCalibration::likelihoodScale, NonDDREAMBayesCalibration::NonDDREAMInstance, Iterator::numContinuousVars, NonDCalibration::numExperiments, Iterator::numFunctions, NonDCalibration::numReplicates, and Iterator::outputLevel.

**13.77.3.5 void quantify_uncertainty () [protected, virtual]**

redefined from DakotaNonD Perform the uncertainty quantification

Reimplemented from NonDBayesCalibration.

References NonDDREAMBayesCalibration::calibrateSigmaFlag, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), NonDBayesCalibration::emulatorModel, NonDDREAMBayesCalibration::emulatorType, NonDCalibration::expData, NonDCalibration::expDataFileAnnotated, NonDCalibration::expDataFileName, Iterator::iterator_rep(), ExperimentData::load_scalar(), NonDDREAMBayesCalibration::NonDDREAMInstance, Iterator::numContinuousVars, NonDCalibration::numExpConfigVars, NonDCalibration::numExperiments, NonDCalibration::numExpStdDeviationsRead, Iterator::numFunctions, NonDCalibration::numReplicates, NonD-
DREAMBayesCalibration::numSamples, Iterator::outputLevel, NonDDREAMBayesCalibration::paramMaxs, NonDDREAMBayesCalibration::paramMins, NonDDREAMBayesCalibration::priorDistributions, NonDDREAMBayesCalibration::priorSamplers, NonDDREAMBayesCalibration::randomSeed, NonDDREAMBayesCalibration::numGenerator, NonDBayesCalibration::standardizedSpace, and NonDBayesCalibration::stochExpIterator.

13.77.4 Member Data Documentation

13.77.4.1 Real likelihoodScale  [protected]

scale factor for proposal covariance scale factor for likelihood
Referenced by NonDDREAMBayesCalibration::sample_likelihood().

The documentation for this class was generated from the following files:

- NonDDREAMBayesCalibration.hpp
- NonDDREAMBayesCalibration.cpp
13.78 NonDExpansion Class Reference

Base class for polynomial chaos expansions (PCE) and stochastic collocation (SC). Inheritance diagram for NonDExpansion::

```
NonDExpansion
  NonD
  NonDPolynomialChaos
  NonDStochCollocation
```

**Public Member Functions**

- **NonDExpansion (Model &model)**
  
  *standard constructor*

- **NonDExpansion (Model &model, short exp_coeffs_approach, short u_space_type, bool piecewise_basis, bool use_derivs)**
  
  *alternate constructor*

- **~NonDExpansion ()**
  
  *destructor*

- **void quantify_uncertainty ()**
  
  *perform a forward uncertainty propagation using PCE/SC methods*

- **void print_results (std::ostream &s)**
  
  *print the final statistics*

- **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*

**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 ()
  form the expansion by calling uSpaceModel.build_approximation()

• virtual void increment_order ()
  uniformly increment the expansion order (PCE only)

• virtual void increment_specification_sequence ()
  increment the input specification sequence (PCE only)

• virtual void update_expansion ()
  update an expansion; avoids overhead in compute_expansion()

• virtual void print_coefficients (std::ostream &s)
  print expansion coefficients, as supported by derived instance

• 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 (short u_space_type)
  common constructor code for initialization of natafTransform

• void refine_expansion ()
  refine the reference expansion found by compute_expansion() using uniform/adaptive p-/h-refinement strategies
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, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  assign a NonDQuadrature instance within u_space_sampler based on a quad_order specification

void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, int filtered_samples, const RealVector &dim_pref)
  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, int random_samples, int seed, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  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, const UShortArray &ssg_level_seq, const RealVector &ssg_dim_pref)
  assign a NonDSparseGrid instance within u_space_sampler

void construct_expansion_sampler ()
  construct the expansionSampler operating on uSpaceModel

void compute_statistics ()
  calculate analytic and numerical statistics from the expansion

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 orthogonal polynomial data fit recursions.

short expansionCoeffsApproach
  method for collocation point generation and subsequent calculation of the expansion coefficients

size_t numUncertainQuant
  number of invocations of quantify_uncertainty()

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_SOBL, DIMENSION_ADAPTIVE_CONTROL_DECAY, or DIMENSION_ADAPTIVE_CONTROL_GENERALIZED

- **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)
• void compute_covariance ()
  calculate the response covariance (diagonal or full matrix)

• void compute_diagonal_variance ()
  calculate respVariance or diagonal terms respCovariance(i,i)

• 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 compute_print_increment_results ()
  manage print of results following a refinement increment

• void compute_print_iteration_results (bool initialize)
  manage print of results following a refinement increment

• void compute_print_converged_results (bool print_override=false)
  manage print of results following convergence of iterative refinement

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.

- bool expSampling
  flag to indicate calculation of numerical statistics by sampling on the expansion

- bool impSampling
  flag to use LHS sampling or MMAIS sampling on the expansion

- RealMatrix expGradsMeanX
  derivative of the expansion with respect to the x-space variables evaluated at the means (used as uncertainty importance metrics)

- bool vbdFlag
  flag indicating the activation of variance-bsaed 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

13.78.1 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.

13.78.2 Member Function Documentation

13.78.2.1 void increment_specification_sequence () [protected, virtual]

increment the input specification sequence (PCE only) default implementation is overridden by PCE
Reimplemented in NonDPolynomialChaos.

References NonDIntegration::increment_specification_sequence(), Iterator::iterator_rep(), Model::subordinate_iterator(), and NonDExpansion::uSpaceModel.
Referenced by NonDExpansion::quantify_uncertainty().
13.78.2.2 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, NonDExpansion::respVariance, and Dakota::write_data().

Referenced by NonDExpansion::increment_sets(), and NonDExpansion::refine_expansion().

13.78.2.3 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::finalStatistics, Response::function_values(), Iterator::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, and NonD::requestedRespLevels.

Referenced by NonDExpansion::increment_sets().

13.78.2.4 void compute_statistics () [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 Dakota::abort_handler(), ResultsManager::active(), Iterator::active_set(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), Iterator::all_responses(), Iterator::all_samples(), Model::approximation_data(), Model::approximations(), NonD::archive_allocate_mappings(), NonDExpansion::archive_coefficients(), NonD::archive_from_resp(), NonDExpansion::archive_moments(), NonD::archive_to_resp(), NonD::cdfFlag, PecosApproximation::compute_component_effects(), PecosApproximation::compute_moments(), NonDExpansion::compute_off_diagonal_covariance(), PecosApproximation::compute_total_effects(), NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRelLevels, NonD::computedRespLevels, Model::continuous_variable_ids(), Model::continuous_variable_labels(), Model::continuous_variables(), Dakota::copy_data(), NonDExpansion::covarianceControl, Model::current_variables(), PecosApproximation::expGradsmeanX, NonDExpansion::expSampling, NonD::finalStatistics, Response::function_gradient(), Response::function_value(), Response::function_values(), NonDAdaptImpSampling::get_probability(), NonDExpansion::importanceSampler, NonDExpansion::impSampling, Iterator::initial_points(), NonDAdaptImpSampling::initialize(), NonD::initialize_distribution_mappings(), NonD::initialize_random_variables(), NonDExpansion::initializePtU, ResultsManager::insert(), Iterator::iteratedModel, Iterator::iterator_rep(), PecosApproximation::mean_gradient(), PecosApproximation::moments(), NonD::natafTransform, NonD::numContDesVars, NonD::numContEpistUncVars, Iterator::numContinuousVars, NonD::numContStateVars, Iterator::numFunctions, NonDExpansion::numSamplesOnExpansion, Iterator::outputLevel, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonDExpansion::respCovariance, NonD::respLevelTarget, Model::response_labels(), Iterator::response_results(), NonDExpansion::respVariance, Iterator::resultsDB, Iterator::resultsNames,
13.78.3 Member Data Documentation

13.78.3.1 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(), NonDStochCollocation::resolve_inputs(), NonDStochCollocation::resolve_inputs(), NonDStochCollocation::resolve_inputs(), NonDPolynomialChaos::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
13.79 NonDGlobalEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ. Inheritance diagram for NonD-GlobalEvidence::

```
   Iterator
      
   Analyzer
      
   NonD
      
   NonDInterval
      
   NonDGlobalInterval
      
   NonDGlobalEvidence
```

Public Member Functions

- **NonDGlobalEvidence (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*
13.79.1 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:

- NonDGlobalEvidence.hpp
- NonDGlobalEvidence.cpp
Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification. Inheritance diagram for NonDGlobalInterval::

Public Member Functions

- **NonDGlobalInterval (Model &model)**
  * constructor *

- **~NonDGlobalInterval ()**
  * destructor *

- **void quantify_uncertainty ()**
  * 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**
  * return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain *

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.

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** `prevFnStar`
  stores previous solution value for assessing convergence

- **size_t** `sbIterNum`
  surrogate-based minimization/maximization iteration count

- **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

### 13.80.1 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.

The documentation for this class was generated from the following files:

- `NonDGlobalInterval.hpp`
- `NonDGlobalInterval.cpp`
13.81 NonDGlobalReliability Class Reference

Class for global reliability methods within DAKOTA/UQ. Inheritance diagram for NonDGlobalReliability::

```
NonDGlobalReliability
  NonDReliability
    NonD
      Analyzer
        Iterator
```

Public Member Functions

- **NonDGlobalReliability (Model &model)**
  constructor

- **~NonDGlobalReliability ()**
  destructor

- **void quantify_uncertainty ()**
  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)**
  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

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

13.81.1 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.

The documentation for this class was generated from the following files:

• NonDGlobalReliability.hpp
• NonDGlobalReliability.cpp
13.82 NonDGlobalInterval 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 (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
13.82.1 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
13.83 NonDGPImpSampling Class Reference

Class for the Gaussian Process-based Importance Sampling method. Inheritance diagram for NonDGPImpSampling:

```
```

Public Member Functions

- **NonDGPImpSampling** (Model &model)
  
  *standard constructor*

- **~NonDGPImpSampling** ()
  
  *alternate constructor for sample generation and evaluation "on the fly"*

- **void quantify_uncertainty** ()
  
  *perform the GP importance sampling and return probability of failure.*

- **const Real & get_probability** ()
  
  *returns the probability calculated by the importance sampling*

- **void print_results** (std::ostream &s)
  
  *print the final statistics*

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

13.83.1 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).

13.83.2 Constructor & Destructor Documentation

13.83.2.1 NonDGPImpSampling (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 Model::assign_rep(), Iterator::assign_rep(), NonD::construct_lhs(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_string(), NonDGPImpSampling::gpBuild, NonDGPImpSampling::gpEval, NonDGPImpSampling::gpModel, Iterator::gradientType, Iterator::hessianType, Model::init_communicators(), Iterator::iteratedModel, Iterator::maximum_concurrency(), Iterator::maximum_iterations, NonDGPImpSampling::numEmulEval, NonDGPImpSampling::numPtsAdd, NonDSampling::numSamples, Iterator::outputLevel, Iterator::probDescDB, NonDSampling::randomSeed, NonDSampling::rngName, NonDGPImpSampling::sampleRhoOne, NonDSampling::samplingVarsMode, NonDSampling::statsFlag, NonDSampling::vary_pattern(), and NonDSampling::varyPattern.
13.83.2.2 ~NonDGPImpSampling ()

alternate constructor for sample generation and evaluation "on the fly" destructor

References Model::free_communicators(), NonDGPImpSampling::gpEval, NonDGPImpSampling::gpModel, and Iterator::maximum_concurrency().

13.83.3 Member Function Documentation

13.83.3.1 void quantify_uncertainty () [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.

Implements NonD.

References Model::acv(), Iterator::all_responses(), Iterator::all_samples(), Analyzer::all_samples(), Model::append_approximation(), Model::approximation_data(), Model::approximation_variances(), Model::build_approximation(), NonDGPImpSampling::calcExpIndicator(), NonDGPImpSampling::calcExpIndPoint(), NonDGPImpSampling::calcRhoDraw(), NonD::cdfFlag, Model::compute_response(), NonD::computedProbLevels, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Model::current_response(), Model::current_variables(), NonDGPImpSampling::drawNewX(), 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_distribution_mappings(), Iterator::iteratedModel, NonDGPImpSampling::normConst, NonDGPImpSampling::numEmulEval, Iterator::numFunctions, NonDGPImpSampling::numPtsAdd, NonDGPImpSampling::numPtsTotal, NonDSampling::numSamples, Iterator::outputLevel, Model::pop_approximation(), NonD::requestedRespLevels, NonDGPImpSampling::rhoDraw, NonDGPImpSampling::rhoDrawThis, NonDGPImpSampling::rhoMix, NonDGPImpSampling::rhoOne, Iterator::run_iterator(), NonDGPImpSampling::sampleRhoOne, and NonDGPImpSampling::xDrawThis.

The documentation for this class was generated from the following files:

- NonDGPImpSampling.hpp
- NonDGPImpSampling.cpp
13.84 NonDGPMSABayesCalibration Class Reference

Generates posterior distribution on model parameters given experiment data. Inheritance diagram for NonDGPMSABayesCalibration::

```
    Iterator
    |    
    v    
Analyzer
    |    
NonD
    |    
NonDCalibration
    |    
NonDBayesCalibration
    |    
NonDGPMSABayesCalibration
```

Public Member Functions

- **NonDGPMSABayesCalibration (Model &model)**
  
  *standard constructor*

- **~NonDGPMSABayesCalibration ()**
  
  *destructor*

Public Attributes

- **String rejectionType**
  
  *Rejection type (standard or delayed, in the DRAM framework).*

- **String metropolisType**
  
  *Metropolis type (hastings or adaptive, in the DRAM framework).*

- **int numSamples**
  
  *number of samples in the chain (e.g. number of MCMC samples)*

- **int emulatorSamples**
  
  *number of samples of the simulation to construct the GP*

- **RealVector proposalCovScale**
  
  *scale factor for proposal covariance*
- Real **likelihoodScale**  
  scale factor for likelihood

- bool **calibrateSigmaFlag**  
  flag to indicated if the sigma terms should be calibrated (default true)

- String **approxImportFile**  
  name of file from which to import build points to build GP

- bool **approxImportAnnotated**  
  annotate flag

**Protected Member Functions**

- void **quantify_uncertainty ()**  
  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.

**Protected Attributes**

- int **randomSeed**  
  print the final statistics

**Private Attributes**

- short **emulatorType**  
  the emulator type: NO_EMULATOR, GAUSSIAN_PROCESS, POLYNOMIAL_CHAOS, or STOCHASTIC_COLLOCATION

- Iterator **lhsIter**  
  LHS iterator for generating samples for GP.

**Static Private Attributes**

- static NonDGPMsABayesCalibration * **NonDGPMsAInstance**  
  Pointer to current class instance for use in static callback functions.
13.84.1 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.

13.84.2 Constructor & Destructor Documentation

13.84.2.1 NonDGPMSABayesCalibration (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 Iterator::assign_rep(), NonDGPMSABayesCalibration::emulatorSamples, ProblemDescDB::get_string(), Model::init_communicators(), Iterator::iteratedModel, NonDGPMSABayesCalibration::lhsIter, Iterator::maximum_concurrency(), Iterator::probDescDB, and NonDGPMSABayesCalibration::randomSeed.

13.84.3 Member Function Documentation

13.84.3.1 void quantify_uncertainty () [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

Reimplemented from NonDBayesCalibration.

References Iterator::all_responses(), Iterator::all_samples(), Analyzer::all_samples(), NonDGPMSABayesCalibration::approxImportAnnotated, NonDGPMSABayesCalibration::approxImportFile, NonDGPMSABayesCalibration::calibrateSigmaFlag, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), NonDGPMSABayesCalibration::emulatorSamples, NonDGPMSABayesCalibration::emulatorType, NonDGPMSABayesCalibration::expData, NonDGPMSABayesCalibration::expDataFileAnnotated, NonDGPMSABayesCalibration::expDataFileName, Iterator::iteratedModel, NonDGPMSABayesCalibration::lhsIter, ExperimentData::load_scalar(), NonDGPMSABayesCalibration::metropolisType, NonDGPMSABayesCalibration::NonDGPMSAInstance, NonDGPMSABayesCalibration::NonDGPMSAInstance, NonD::numExpConfigVars, NonD::numExperiments, NonD::numExpStdDeviationsRead, Iterator::numFunctions, NonD::numReplicates, NonD::numSamples, NonD::numUncertainVars, Iterator::outputLevel, Dakota::read_data_tabular(), NonDGPMSABayesCalibration::rejectionType, and Iterator::run_iterator().
13.84.4 Member Data Documentation

13.84.4.1 int randomSeed [protected]

print the final statistics random seed to pass to QUESO

Referenced by NonDGPMSABayesCalibration::NonDGPMSABayesCalibration().

The documentation for this class was generated from the following files:

- NonDGPMSABayesCalibration.hpp
- NonDGPMSABayesCalibration.cpp
13.85 NonDIncremLHSSampling Class Reference

Performs incremental LHS sampling for uncertainty quantification. Inheritance diagram for NonDIncremLHSSampling::

```
NonDIncremLHSSampling
\downarrow
NonDSampling
\downarrow
NonD
\downarrow
Analyzer
\downarrow
Iterator
```

Public Member Functions

- **NonDIncremLHSSampling** (Model &model)
  
  *constructor*

- **\sim NonDIncremLHSSampling** ()
  
  *destructor*

- **void quantify_uncertainty** ()
  
  *performs a forward uncertainty propagation by using LHS to generate a set of parameter samples, performing
  function evaluations on these parameter samples, and computing statistics on the ensemble of results.*

- **void print_results** (std::ostream &s)
  
  *print the final 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

- **int previousSamples**
  
  *number of samples in previous LHS run*
13.85 NonDIncremLHSSampling Class Reference

- bool varBasedDecompFlag
  
  flags computation of VBD

Static Private Attributes

- static RealArray rawData
  
  static data used by static rank_sort() fn

13.85.1 Detailed Description

Performs incremental LHS 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. 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.

13.85.2 Constructor & Destructor Documentation

13.85.2.1 NonDIncremLHSSampling (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.

13.85.3 Member Function Documentation

13.85.3.1 void quantify_uncertainty () [virtual]

performs a forward uncertainty propagation by using LHS to generate a set of parameter samples, performing function evaluations on these parameter samples, and computing statistics on the ensemble of results. Generate incremental samples. Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.

Implements NonD.

References Dakota::abort_handler(), Model::aleatory_distribution_parameters(), Analyzer::allResponses, Analyzer::allSamples, NonDSampling::compute_statistics(), Dakota::copy_data(), Dakota::data_pairs, Analyzer::evaluate_parameter_sets(), NonDSampling::get_parameter_sets(), Iterator::iteratedModel, NonD::numBetaVars, NonD::numBinomialVars, Iterator::numContinuousVars, NonD::numExponentialVars, NonD::numFrechetVars, NonD::numGammaVars, NonD::numGeometricVars, NonD::numGumbelVars, NonD::numHistogramBinVars, NonD::numHistogramPtVars, NonD::numHyperGeomVars, NonD::numLognormalVars, NonD::numLoguniformVars, NonD::numNegBinomialVars, NonD::numNormalVars, NonD::numPoissonVars, NonDSampling::numSamples, NonD::numTriangularVars, NonD::numUniformVars, NonD::numWeibullVars, NonD::numIncremLHSSampling::previousSamples,
NonDIncremLHSSampling::rank_sort(), NonDIncremLHSSampling::rawData, NonDSampling::sampleRanks,
NonDSampling::sampleRanksMode, NonDSampling::samplesRef, NonDSampling::samplesRef,
Dakota::strends(), NonDSampling::varyPattern, and Dakota::write_data().

The documentation for this class was generated from the following files:

- NonDIncremLHSSampling.hpp
- NonDIncremLHSSampling.cpp
13.86 NonDIntegration Class Reference

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

```
NonDIntegration
    NonD
    Analyzer
    Iterator
    NonDCubature
    NonDQuadrature
    NonDSparseGrid
```

**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*

- virtual void `increment_specification_sequence` ()
  
  *increment specificationSequence and update active orders/levels*

- const Pecos::IntegrationDriver & `driver` () const
  
  *return numIntDriver*

**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 (Model &model)
  constructor

• NonDIntegration (NoDBBaseConstructor, Model &model)
  alternate constructor for instantiations "on the fly"

• NonDIntegration (NoDBBaseConstructor, Model &model, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly"

• ~NonDIntegration ()
  destructor

• void quantify_uncertainty ()
  performs a forward uncertainty propagation of parameter distributions into response statistics

• void check_variables (const Pecos::ShortArray &x_types)
  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

• size_t sequenceIndex
  index into NonDQuadrature::quadOrderSpec and NonDSparseGrid::ssgLevelSpec that defines the current instance of several possible refinement levels

• RealVector dimPrefSpec
  the user specification for anisotropic dimension preference
13.86 NonDIntegration Class Reference

13.86.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals. This class provides a base class for shared code among NonDQuadrature and NonDSparseGrid.

13.86.2 Constructor & Destructor Documentation

13.86.2.1 NonDIntegration (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_quadrature/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(), Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, and NonD::verify_correlation_support().

13.86.2.2 NonDIntegration (NoDBBaseConstructor, 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.

13.86.2.3 NonDIntegration (NoDBBaseConstructor, 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.

13.86.3 Member Function Documentation

13.86.3.1 void dimension_preference_to_anisotropic_order (unsigned short scalar_order_spec, const RealVector & dim_pref_spec, size_t num_v, UShortArray & aniso_order) [static]

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::increment_specification_sequence(), NonDQuadrature::initialize_dimension_quadrature_order(), and NonDPolynomialChaos::NonDPolynomialChaos().

13.86.3.2 void anisotropic_order_to_dimension_preference (const UShortArray & aniso_order, unsigned short & scalar_order, RealVector & dim_pref) [static]

Converts a vector anisotropic order into a scalar order and vector anisotropic dimension preference.

Referenced by NonDPolynomialChaos::NonDPolynomialChaos().
13.86.3.3 void check_variables (const Pecos::ShortArray & x_types)  [protected]

verify self-consistency of variables data Virtual function called from probDescDB-based constructors and from NonDIntegration::quantify_uncertainty()

References Dakota::abort_handler(), NonD::numContAleatUncVars, NonD::numContDesVars, NonD::numContEpistUncVars, Iterator::numContinuousVars, and NonD::numContStateVars.

Referenced by NonDCubature::NonDCubature(), NonDQuadrature::NonDQuadrature(), NonDSparseGrid::NonDSparseGrid(), and NonDIntegration::quantify_uncertainty().

The documentation for this class was generated from the following files:

- NonDIntegration.hpp
- NonDIntegration.cpp
13.87 NonDInterval Class Reference

Base class for interval-based methods within DAKOTA/UQ. Inheritance diagram for NonDInterval::

```
NonDInterval
| NonDInterval
| NonDAnalyzer
| NonDIterator

NonDGlobalInterval NonDLocalInterval

NonDGlobalEvidence NonDLocalEvidence
NonDGlobalSingleInterval NonDLocalSingleInterval
NonDLHSEvidence NonDLHSSingleInterval
```

Public Member Functions

- **NonDInterval (Model &model)**
  
  *constructor*

- **~NonDInterval ()**
  
  *destructor*

- **void print_results (std::ostream &s)**

  *performs an epistemic uncertainty propagation using Dempster-Shafer evidence theory methods which solve for cumulative distribution functions of belief and plausibility*

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.

- `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.

- `RealVectorArray cellBPA`
  Storage array to hold cell bpa.

- `size_t respFnCntr`
  response function counter
13.87 NonDInterval Class Reference

- size_t cellCntr
  
  cell counter

- size_t numCells
  
  total number of interval combinations

13.87.1 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.

13.87.2 Member Function Documentation

13.87.2.1 void print_results (std::ostream & s) [virtual]

performs an epistemic uncertainty propagation using Dempster-Shafer evidence theory methods which solve for cumulative distribution functions of belief and plausibility print the cumulative distribution functions for belief and plausibility

Reimplemented from Analyzer.

References NonDInterval::ccBelFn, NonDInterval::ccBelVal, NonDInterval::ccPlausFn, NonD-Interval::ccPlausVal, NonD::cdfFlag, NonDInterval::cellBPA, NonDInterval::cellFnLowerBounds, NonInterval::cellFnUpperBounds, NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRespLevels, NonD::finalStatistics, Response::function_values(), Iterator::iteratedModel, NonDInterval::numCells, Iterator::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRespLevels, NonD::respLevelTarget, Model::response_labels(), NonDInter-val::singleIntervalFlag, and Dakota::write_precision.

The documentation for this class was generated from the following files:

- NonDInterval.hpp
- NonDInterval.cpp
13.88 NonDLHSEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ. Inheritance diagram for NonDLHSEvidence::

```plaintext
NonDLHSEvidence
  Iterator
    Analyzer
      NonD
        NonDInterval
          NonDLHSInterval
            NonDLHSEvidence
```

Public Member Functions

- **NonDLHSEvidence (Model &model)**
  
  *constructor*

- **~NonDLHSEvidence ()**
  
  *destructor*

- **void initialize ()**
  
  *perform any required initialization*

- **void post_process_samples ()**
  
  *post-process the output from executing lhsSampler*

13.88.1 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
13.89 NonDLHSInterval Class Reference

Class for the LHS-based interval methods within DAKOTA/UQ. Inheritance diagram for NonDLHSInterval:

![Inheritance Diagram]

Public Member Functions

- **NonDLHSInterval (Model &model)**
  
  constructor

- **~NonDLHSInterval ()**
  
  destructor

- **void quantify_uncertainty ()**
  
  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

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*

### 13.89.1 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
13.90 NonDLHSSampling Class Reference

Performs LHS and Monte Carlo sampling for uncertainty quantification. Inheritance diagram for NonDLHSSampling::

```
NonDLHSSampling
   NonDSampling
      NonD
         Analyzer
            Iterator
```

**Public Member Functions**

- `NonDLHSSampling (Model &model)`  
  *standard constructor*

- `NonDLHSSampling (Model &model, const String &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 (const String &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"*

- `~NonDLHSSampling ()`  
  *destructor*

**Protected Member Functions**

- `void pre_run ()`  
  *generate LHS samples in non-VBD cases*

- `void post_input ()`  
  *read tabular data for post-run mode*

- `void quantify_uncertainty ()`  
  *perform the evaluate parameter sets portion of run*
13.90 NonDLHSSampling Class Reference

- void **post_run** (std::ostream &s)
  
  *generate statistics for LHS runs in non-VBD cases*

- void **print_results** (std::ostream &s)
  
  *print the final statistics*

**Private Attributes**

- size_t **numResponseFunctions**
  
  *number of response functions; used to distinguish NonD from opt/NLS usage*

- bool **varBasedDecompFlag**
  
  *flags computation of variance-based decomposition indices*

13.90.1  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.

13.90.2  Constructor & Destructor Documentation

13.90.2.1  NonDLHSSampling (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.

13.90.2.2  NonDLHSSampling (Model & model, const String & 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.
13.90.2.3 NonDLHSSampling (const String & 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" 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().

13.90.3 Member Function Documentation

13.90.3.1 void quantify_uncertainty () [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.

Implements NonD.

References NonDSampling::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, Iterator::numContinuousVars, Iterator::numDiscreteIntVars, Iterator::numDiscreteRealVars, NonDLHSSampling::numResponseFunctions, NonDSampling::numSamples, NonDSampling::statsFlag, NonDLHSSampling::varBasedDecompFlag, and Analyzer::variance_based_decomp().

The documentation for this class was generated from the following files:

- NonDLHSSampling.hpp
- NonDLHSSampling.cpp
Class for pure interval propagation using LHS. Inheritance diagram for NonDLHSSingleInterval:

![Inheritance Diagram]

### Public Member Functions

- **NonDLHSSingleInterval (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*
13.91.1 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
13.92 NonDLocalEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ. Inheritance diagram for NonDLocalEvidence:

```
           Iterator
           |
         Analyzer
         |
           NonD
         |
     NonDInterval
         |
  NonDLocalInterval
         |
NonDLocalEvidence
```

**Public Member Functions**

- **NonDLocalEvidence (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

13.92.1 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
13.93 NonDLocalInterval Class Reference

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

```
Iterator
   Analyzer
      NonD
         NonDInterval
            NonDLocalInterval
                NonDLocalEvidence NonDLocalSingleInterval
```

Public Member Functions

- **NonDLocalInterval (Model &model)**
  
  *constructor*

- **~NonDLocalInterval ()**
  
  *destructor*

- **void quantify_uncertainty ()**
  
  *Performs an optimization to determine interval bounds for an entire function or interval bounds on a particular statistical estimator.*

- **String 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 * `nondLIInstance`
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
13.93.1 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
13.94 NonDLocalReliability Class Reference

Class for the reliability methods within DAKOTA/UQ. Inheritance diagram for NonDLocalReliability:

```
    Iterator
     |
    Analyzer
     |
    NonD
     |
NonDReliability
     |
NonDLocalReliability
```

**Public Member Functions**

- **NonDLocalReliability (Model &model)**
  *constructor*

- **~NonDLocalReliability ()**
  *destructor*

- **void quantify_uncertainty ()**
  *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)**
  *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*

- **String 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 update_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_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 \( \| u \|^2 = (\beta - \beta) \).

• 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 \( \beta_* = \beta_* - \beta \).

• 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 ranVarMeansU
  vector of means for all uncertain random variables in u-space

• RealVector initialPtU
  initial guess for MPP search 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 hessianType

• 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
13.94.1 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/MVSSOM), advanced mean value method (AMV, AMV^2) in x- or u-space, iterated advanced mean value method (AMV+, AMV^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.

13.94.2 Member Function Documentation

13.94.2.1 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)^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().

13.94.2.2 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().

13.94.2.3 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.
References Response::active_set_request_vector(), Variables::continuous_variables(), NonDLocalReliability::curvatureDataAvailable, Response::function_gradient(), Response::function_gradient_view(), Response::function_hessian(), Response::function_hessian_view(), Response::function_value(), NonDLocalReliability::integrationOrder, NonDLocalReliability::kappaUpdated, NonDReliability::mppSearchType, NonDLocalReliability::nondLocRelInstance, NonDReliability::pmaMaximizeG, NonDReliability::respFnCount, NonDLocalReliability::update_pma_maximize(), and Dakota::write_data).

Referenced by NonDLocalReliability::mpp_search().

13.94.2.4 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 \( \|u\|^2 = (\beta - \bar{\beta})^2 \). 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 Response::active_set_request_vector(), Variables::continuous_variables(), Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), NonDLocalReliability::nondLocRelInstance, and NonDReliability::requestedTargetLevel.

Referenced by NonDLocalReliability::mpp_search().

13.94.2.5 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 \( \beta^* = \beta^* - \bar{\beta} \). 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(), Response::function_value(), NonDLocalReliability::mostProbPointU, NonDReliability::mppSearchType, NonDLocalReliability::nondLocRelInstance, NonDLocalReliability::probability(), NonDLocalReliability::reliability(), NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, NonDLocalReliability::signed_norm(), and Dakota::write_data().

Referenced by NonDLocalReliability::mpp_search().

13.94.2.6 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::compute_response(), Model::continuous_variables(), Model::current_response(), NonD::finalStatistics,
NonDLocalReliability::fnGradsMeanX, NonDLocalReliability::fnHessiansMeanX, NonDLocalReliability::fnValsMeanX, Response::function_gradients(), Response::function_hessians(), Response::function_values(), Iterator::hessianType, Iterator::iteratedModel, NonD::momentStats, NonDReliability::mppSearchType, NonD::natafTransform, Iterator::numFunctions, NonD::numUncertainVars, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, Iterator::subIteratorFlag, NonDLocalReliability::taylorOrder, and NonDReliability::uSpaceModel.

Referenced by NonDLocalReliability::quantify_flag().

13.94.2.7 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.

References Response::active_set_derivative_vector(), NonD::finalStatistics, NonDReliability::importanceSampler, NonD::initialize_random_variables(), NonDReliability::integrationRefinement, Iterator::iterator_rep(), NonDReliability::mppModel, NonD::natafTransform, Iterator::numFunctions, NonDReliability::numRelAnalyses, NonD::numUncertainVars, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevGradDLev0, NonDLocalReliability::prevGradULev0, NonDLocalReliability::prevMPPULev0, NonDLocalReliability::ranVarMeansU, Iterator::subIteratorFlag, Model::update_from_subordinate_model(), and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

13.94.2.8 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).

References Iterator::activeSet, NonDLocalReliability::assign_mean_data(), Model::component_parallel_mode(), Model::compute_response(), NonDLocalReliability::computedRespLevel, Model::continuous_variable_ids(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::fnGradU, NonDLocalReliability::fnGradX, NonDLocalReliability::fnHessU, NonDLocalReliability::fnHessX, Response::function_gradient_copy(), Response::function_hessian(), Response::function_value(), Model::inactive_continuous_variables(), NonDLocalReliability::initialPtU, Iterator::iteratedModel, NonDLocalReliability::kappaUpdated, NonDLocalReliability::mostProbPointU, NonDLocalReliability::mostProbPointX, NonDReliability::mppSearchType, NonD::natafTransform, NonDReliability::numRelAnalyses, NonD::numUncertainVars, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevGradDLev0, NonDLocalReliability::prevGradULev0, NonDLocalReliability::prevICVars, NonDLocalReliability::prevMPPULev0, 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().
13.94.2.9 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, Iterator::hessianType, NonDLocalReliability::initialPtU, NonDLocalReliability::integrationOrder, NonDReliability::levelCount, NonDLocalReliability::mostProbPointU, NonDReliability::mppSearchType, NonD::numUncertainVars, NonD::requestedProbLevels, NonD::requestedRelLevels, NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, NonDLocalReliability::taylorOrder, and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

13.94.2.10 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.

References Response::active_set(), Response::active_set_request_vector(), Iterator::activeSet, NonDReliability::approxConverged, NonDReliability::approxIters, Model::component_parallel_mode(), Model::compute_response(), NonDLocalReliability::computedRelLevel, NonDLocalReliability::computedRespLevel, Model::continuous_variable_ids(), Model::continuous_variables(), Variables::continuous_variables(), Iterator::convergenceTol, Variables::copy(), Dakota::copy_data(), Model::current_response(), Model::current_variables(), NonDLocalReliability::curvatureDataAvailable, Dakota::data_pairs, NonD::finalStatistics, NonDLocalReliability::fnGradU, NonDLocalReliability::fnGradX, NonDLocalReliability::fnHessU, NonDLocalReliability::fnHessX, Response::function_gradient_copy(), Response::function_hessian(), Response::function_value(), Response::function_values(), NonDLocalReliability::initialPtU, NonDLocalReliability::integrationOrder, Model::interface_id(), Iterator::iteratedModel, NonDLocalReliability::kappaUpdated, NonDReliability::levelCount, Dakota::lookup_by_val(), Iterator::maxIterations, NonDLocalReliability::mostProbPointU, NonDLocalReliability::mostProbPointX, NonDReliability::mppSearchType, NonD::natafTransform, Iterator::numFunctions, NonD::numNormalVars, NonD::numUncertainVars, NonDReliability::pmaMaximizeG, ActiveSet::request_value(), ActiveSet::request_values(), ActiveSet::request_vector(), NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, NonDLocalReliability::taylorOrder, NonDLocalReliability::update_limit_state_surrogate(), NonDLocalReliability::update_pma_maximize(), NonDReliability::uSpaceModel, NonDLocalReliability::warmStartFlag, and NonDLocalReliability::warningBits.

Referenced by NonDLocalReliability::mpp_search().

13.94.2.11 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.

References Response::active_set_derivative_vector(), Response::active_set_request_vector(), Graphics::add_
13.94.2.12 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(), Model::compute_response(), Dakota::contains(), Model::continuous_variable_ids(), Model::continuous_variables(), Dakota::copy_data(), Model::current_response(), ActiveSet::derivative_vector(), NonD::finalStatistics, Response::function_gradient_copy(), Response::function_gradients(), Model::inactive_continuous_variable_ids(), Iterator::iteratedModel, NonDReliability::mppSearchType, NonD::natafTransform, 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().

13.94.2.13 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().
13.94.2.14 Real probability (Real \( \beta \), bool \( \text{cdf\_flag} \), const RealVector & \( \text{mpp\_u} \), const RealVector & \( \text{fn\_grad\_u} \), const RealSymMatrix & \( \text{fn\_hess\_u} \)) [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::get_probability(), NonDReliability::importanceSampler, NonDLocalReliability::integrationOrder, NonDReliability::integrationRefinement, Iterator::iterator_rep(), NonDLocalReliability::kappaU, NonDLocalReliability::kappaUpdated, NonD::numUncertainVars, Iterator::outputLevel, NonDLocalReliability::principal_curvatures(), NonDLocalReliability::probability(), NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, Iterator::run_iterator(), NonDLocalReliability::scale_-curvature(), NonDLocalReliability::secondOrderIntType, NonDLocalReliability::warningBits, Dakota::write_-data(), and Dakota::write_precision.

The documentation for this class was generated from the following files:

- NonDLocalReliability.hpp
- NonDLocalReliability.cpp
13.95 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 (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"
13.95.1 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 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:

- NonDLocalSingleInterval.hpp
- NonDLocalSingleInterval.cpp
13.96 NonDPOFDarts Class Reference

Base class for POF Dart methods within DAKOTA/UQ. Inheritance diagram for NonDPOFDarts:

```
NonDPOFDarts
  |         |
  |         |
  |         |
  NonD    NonDPOFDarts
  |         |
  |         |
  Iterator
  |         |
  Analyzer
```

**Public Member Functions**

- **NonDPOFDarts (Model &model)**
  
  *constructor*

- **~NonDPOFDarts ()**
  
  *destructor*

- **void quantify_uncertainty ()**
  
  *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_POF_results (double lower, double upper)**

- **void print_results (std::ostream &s)**

  *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 shrink_big_spheres()
• void assign_sphere_radius_POF (double *x, size_t isample)
• void assign_sphere_radius_OPT (double *x, size_t isample)
• void resolve_overlap_POF (size_t ksample)
• void compute_response (double *x)
• void compute_response_for_FD_gradients (double *x)
• double get_dart_radius (double f, double fgrad, double fcurv)
• void retrieve_POF_bounds (double &lower, double &upper)
• double estimate_spheres_volume_0d (double **spheres, size_t num_spheres, size_t num_dim, double *xmin, double *xmax)
• double get_sphere_volume (double r, size_t num_dim)
• void sample_uniformly_from_unit_sphere (double *dart, size_t num_dim)
• double area_triangle (double x1, double y1, double x2, double y2, double x3, double y3)
• double f_true (double *x)
• void plot_vertices_2d ()

Protected Attributes

• int samples
• int seed
• double Q [1220]
• int indx
• double cc
• double c
• double zc
• double zx
• double zy
• size_t qlen
• size_t _n_dim
• double * _xmin
• double * _xmax
• bool _global_optimization
• double _failure_threshold
• double _global_minima
• double _num_darts
• double _num_successive_misses_p
• double _num_successive_misses_m
• double _max_num_successive_misses
• double _max_radius
• double _accepted_void_ratio
• size_t _num_inserted_points
• size_t _total_budget
• double ** _sample_points
• double * _dart
• double * _grad_vec
• 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 ** _fval
• size_t _ieval
• size_t _active_response_function
• double _dx
• size_t _num_sample_eval

13.96.1 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
13.97 NonDPolynomialChaos Class Reference

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification. Inheritance diagram for NonDPolynomialChaos::

```
    Iterator
      Analyzer
        NonD
          NonDExpansion
            NonDPolynomialChaos
```

Public Member Functions

- `NonDPolynomialChaos (Model &model)`
  `standard constructor`

- `NonDPolynomialChaos (Model &model, short exp_coeffs_approach, unsigned short num_int_level, short u_space_type, bool piecewise_basis, bool use_derivs)`
  `alternate constructor`

- `~NonDPolynomialChaos ()`
  `destructor`

Protected Member Functions

- `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 increment_specification_sequence ()`
  `increment the input specification sequence (PCE only)`

- `void compute_expansion ()`
  `form or import an orthogonal polynomial expansion using PCE methods`
• void `increment_order()`
  
  *uniformly increment the order of the polynomial chaos expansion*

• void `print_coefficients` (std::ostream &s)
  
  *print the PCE coefficient array for the orthogonal basis*

• void `archive_coefficients()`
  
  *archive the PCE coefficient array for the orthogonal basis*

### Private Member Functions

• `int terms_ratio_to_samples(size_t num_exp_terms, Real colloc_ratio, Real terms_order)`
  
  *convert 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, Real terms_order)`
  
  *convert number of expansion terms and number of collocation samples to a collocation ratio*

• 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

• String `expansionImportFile`
  
  *filename for import of chaos coefficients*

• Real `collocRatio`
  
  *factor applied to terms' termsOrder in computing number of regression points, either user specified or inferred*

• Real `termsOrder`
  
  *exponent applied to number of expansion terms for computing number of regression points*

• `bool tensorRegression`
  
  *option for regression PCE using a filtered set tensor-product points*

• `bool crossValidation`
  
  *flag for use of cross-validation for selection of parameter settings in regression approaches*

• `RealVector noiseTols`
  
  *noise tolerance for compressive sensing algorithms; vector form used in cross-validation*

• Real `L2Penalty`
  
  *L2 penalty for LASSO algorithm (elastic net variant).*
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.

### 13.97.1 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.

### 13.97.2 Constructor & Destructor Documentation

#### 13.97.2.1 NonDPolynomialChaos (Model & model)

standard constructor This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.
ProblemDescDB::get_usa(), ProblemDescDB::get_ushort(), Model::init_communicators(), NonDExpansion::initialize(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::maxConcurrency, NonD::numContDesVars, NonD::numContEpistUncVars, Iterator::numContinuousVars, NonD::numContStateVars, NonDExpansion::numSamplesOnExpansion, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::probDescDB, NonDExpansion::refineControl, NonDExpansion::refineType, NonDPolynomialChaos::resolve_inputs(), NonDPolynomialChaos::sequenceIndex, NonDPolynomialChaos::tensorRegression, NonDPolynomialChaos::terms_ratio_to_samples(), NonDPolynomialChaos::terms_samples_to_ratio(), NonDPolynomialChaos::termsOrder, NonD::transform_model(), NonDExpansion::uSpaceModel, and Analyzer::vary_pattern.

13.97.2.2 NonDPolynomialChaos (Model & model, short exp_coeffs_approach, unsigned short num_int_level, short u_space_type, bool piecewise_basis, bool use_derivs)

alternate constructor This constructor is used for helper iterator instantiation on the fly.

References Model::assign_rep(), NonDExpansion::construct_cubature(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), NonDExpansion::expansionCoeffsApproach, NonDExpansion::initialize(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Iterator::outputLevel, NonDPolynomialChaos::resolve_inputs(), NonD::transform_model(), and NonDExpansion::uSpaceModel.

13.97.3 Member Function Documentation

13.97.3.1 void increment_specification_sequence () [protected, virtual]

increment the input specification sequence (PCE only) default implementation is overridden by PCE

Reimplemented from NonDExpansion.

References Model::approximations(), NonDPolynomialChaos::collocPtsSeqSpec, NonDPolynomialChaos::collocRatio, NonDIntegration::dimension_preference_to_anisotropic_order(), NonDPolynomialChaos::dimPrefSpec, PecosApproximation::expansion_order(), NonDExpansion::expansionCoeffsApproach, NonDPolynomialChaos::expOrderSeqSpec, NonDPolynomialChaos::expOrderSeqSpec, Iterator::iterator_rep(), NonDQuadrature::mode(), Model::model_rep(), Iterator::numContinuousVars, Iterator::numFunctions, NonDExpansion::numSamplesOnModel, NonDQuadrature::quadrature_order(), NonDQuadrature::samples(), NonDSampling::sampling_reference(), NonDPolynomialChaos::sequenceIndex, NonDPolynomialChaos::sequenceIndex, Model::subordinate_iterator(), NonDPolynomialChaos::tensorRegression, NonDPolynomialChaos::terms_ratio_to_samples(), NonDPolynomialChaos::termsOrder, DataFitSurrModel::total_points(), NonDQuadrature::update(), PecosApproximation::update_order(), and NonDExpansion::uSpaceModel.

13.97.3.2 void increment_order () [protected, virtual]

uniformly increment the order of the polynomial chaos expansion Used for uniform refinement of regression-based PCE.

Reimplemented from NonDExpansion.

References Model::approximations(), NonDPolynomialChaos::collocRatio, PecosApproximation::expansion_terms(), NonDQuadrature::increment_grid(), PecosApproximation::increment_order(), Iterator::iterator_rep(), NonDQuadrature::mode(), Model::model_rep(), Iterator::numFunctions, NonDExpansion::...
The documentation for this class was generated from the following files:

- NonDPolynomialChaos.hpp
- NonDPolynomialChaos.cpp
13.98 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::

```
  NonDQuadrature
  NonDIntegration
  NonD
  Analyzer
  Iterator
```

Public Member Functions

- **NonDQuadrature** (Model &model, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly" based on a quadrature order specification

- **NonDQuadrature** (Model &model, int num_filt_samples, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly" that generate a filtered tensor product sample set

- **NonDQuadrature** (Model &model, int num_rand_samples, int seed, const UShortArray &quad_order_seq, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly" that sample 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

- 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 **samples** (size_t samples)
  set numSamples
• short mode () const
  
  return quadMode

Protected Member Functions

• NonDQuadrature (Model &model)
  
  constructor

• ~NonDQuadrature ()
  
  destructor

• void initialize_grid (const std::vector<Pecos::BasisPolynomial>& poly_basis)
• void get_parameter_sets (Model &model)
  
  Returns one block of samples (ndim * num_samples).

• void reset ()
  
  restore initial state for repeated sub-iterator executions

• 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

• void increment_specification_sequence ()
  
  increment sequenceIndex and update active orders/levels

• int num_samples () const
  
  get the current number of samples

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

• UShortArray quadOrderSeqSpec
  a sequence of scalar quadrature orders, one per refinement level

• 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

13.98.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation 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 OrthogonalPolynomial class and are extended to n-dimensions using a tensor product approach.

### 13.98.2 Constructor & Destructor Documentation

#### 13.98.2.1 NonDQuadrature (Model & model, const UShortArray & quad_order_seq, const RealVector & dim_pref)

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.

#### 13.98.2.2 NonDQuadrature (Model & model, int num_filt_samples, const RealVector & dim_pref)

alternate constructor for instantiations "on the fly" that generate a filtered tensor product sample set This alternate constructor is used for on-the-fly generation and evaluation of filtered tensor quadrature points. References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

#### 13.98.2.3 NonDQuadrature (Model & model, int num_rand_samples, int seed, const UShortArray & quad_order_seq, const RealVector & dim_pref)

alternate constructor for instantiations "on the fly" that sample 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.

#### 13.98.2.4 NonDQuadrature (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(), ProblemDescDB::get_bool(), ProblemDescDB::get_short(), Iterator::maxConcurrency, NonD::natafTransform, NonDQuadrature::nestedRules, NonDIntegration::numIntDriver, Iterator::probDescDB, NonDQuadrature::reset(), and NonDQuadrature::tpqDriver.

### 13.98.3 Member Function Documentation

#### 13.98.3.1 void initialize_grid (const std::vector< Pecos::BasisPolynomial > & poly_basis) [protected, virtual]

Used in combination with alternate NonDQuadrature constructor.
Implements `NonDIntegration`.

References `Iterator::numContinuousVars`, `NonDQuadrature::nestedRules`, `NonDQuadrature::numSamples`, `NonDQuadrature::quadMode`, `NonDQuadrature::reset()`, `NonDQuadrature::tpqDriver`, and `NonDQuadrature::update()`.

### 13.98.3.2 `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`, `Iterator::numContinuousVars`, and `NonDQuadrature::tpqDriver`.

Referenced by `NonDQuadrature::update()`.

### 13.98.3.3 `int num_samples () const` [inline, protected, virtual]

get the current number of samples 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 maxConcurrency. May be (is) overridden by derived classes.

Reimplemented from `Iterator`.

References `NonDQuadrature::numSamples`, `NonDQuadrature::quadMode`, and `NonDQuadrature::tpqDriver`.

The documentation for this class was generated from the following files:

- `NonDQuadrature.hpp`
- `NonDQuadrature.cpp`
13.99 NonDQUESOBayesCalibration Class Reference

Bayesian inference using the QUESO library from UT Austin. Inheritance diagram for NonDQUESOBayesCalibration::

```
NonDQUESOBayesCalibration
  NonDBayesCalibration
    NonDCalibration
      NonD
        Iterator
        Analyzer
```

Public Member Functions

- **NonDQUESOBayesCalibration** (Model &model)
  
  *standard constructor*

- **~NonDQUESOBayesCalibration** ()

  *destructor*

Public Attributes

- String `rejectionType`
  
  *Rejection type (standard or delayed, in the DRAM framework).*

- String `metropolisType`
  
  *Metropolis type (hastings or adaptive, in the DRAM framework).*

- int `numSamples`
  
  *number of samples in the chain (e.g. number of MCMC samples)*

- RealVector `proposalCovScale`
  
  *scale factor for proposal covariance*

- Real `likelihoodScale`
  
  *scale factor for likelihood*
● bool calibrateSigmaFlag

    flag to indicated if the sigma terms should be calibrated (default true)

Protected Member Functions

● void quantify_uncertainty ()

    redefined from DakotaNonD

Static Protected Member Functions

● static double dakotaLikelihoodRoutine (const uqGslVectorClass &paramValues, const uqGslVectorClass *paramDirection, const void *functionDataPtr, uqGslVectorClass *gradVector, uqGslMatrixClass *hessianMatrix, uqGslVectorClass *hessianEffect)

    Likelihood function for call-back from QUESO to DAKOTA for evaluation.

Protected Attributes

● int randomSeed

    random seed to pass to QUESO

Private Attributes

● short emulatorType

    the emulator type: NO_EMULATOR, GAUSSIAN_PROCESS, POLYNOMIAL_CHAOS, or STOCHASTIC_COLLOCATION

Static Private Attributes

● static NonDQUESOBayesCalibration * NonDQUESOInstance

    Pointer to current class instance for use in static callback functions.

13.99.1 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.
13.99.2 Constructor & Destructor Documentation

13.99.2.1 NonDQUESOBayesCalibration (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.

13.99.3 Member Function Documentation

13.99.3.1 void quantify_uncertainty () [protected, virtual]

redefined from DakotaNonD Perform the uncertainty quantification
Reimplemented from NonDBayesCalibration.
References NonDQUESOBayesCalibration::calibrateSigmaFlag, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), NonDQUESOBayesCalibration::dakotaLikelihoodRoutine(), NonDBayesCalibration::emulatorModel, NonDQUESOBayesCalibration::emulatorType, NonDCalibration::expData, NonDCalibration::expDataFileAnnotated, NonDCalibration::expDataFileName, Iterator::iterator_rep(), ExperimentData::load_scalar(), NonDQUESOBayesCalibration::metropolisType, NonDQUESOBayesCalibration::NonDQUESOInstance, Iterator::numContinuousVars, NonDCalibration::numExpConfigVars, NonDCalibration::numExperiments, NonDCalibration::numExpStdDeviationsRead, Iterator::numFunctions, NonDCalibration::numReplicates, NonDQUESOBayesCalibration::numSamples, Iterator::outputLevel, NonDQUESOBayesCalibration::proposalCovScale, NonDQUESOBayesCalibration::randomSeed, NonDQUESOBayesCalibration::rejectionType, NonDBayesCalibration::standardizedSpace, NonDBayesCalibration::stochExpIterator, and Dakota::strends().

The documentation for this class was generated from the following files:

- NonDQUESOBayesCalibration.hpp
- NonDQUESOBayesCalibration.cpp
13.100  NonDReliability Class Reference

Base class for the reliability methods within DAKOTA/UQ. Inheritance diagram for NonDReliability::

```
NonDReliability
    NonD
    Analyzer
    Iterator
```

Protected Member Functions

- **NonDReliability** (*Model* &model)
  
  *constructor*

- **~NonDReliability** ()
  
  *destructor*

- void **initialize_graphics** (bool graph_2d, bool tabular_data, const *String* &tabular_file)
  
  *initialize graphics customized for reliability methods*

- 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*

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.*

- 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.APPROX

- Iterator importanceSampler
  importance sampling instance used to compute/refine probabilities

- short integrationRefinement
  integration refinement type (NO_INT_REFINE, IS, AIS, or MMAIS) provided by refinement specification

- size_t numRelAnalyses
  number of invocations of quantify_uncertainty()

- 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

### 13.100.1 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.

The documentation for this class was generated from the following files:

- NonDReliability.hpp
- NonDReliability.cpp
13.101 NonDSampling Class Reference

Base class for common code between NonDLHSSampling, NonDIncremLHSSampling, and NonDAdaptImpSampling. Inheritance diagram for NonDSampling::<br>

![Inheritance Diagram for NonDSampling]

**Public Member Functions**

- void `compute_distribution_mappings` (const IntResponseMap &samples)<br>
  *called by compute_statistics() to calculate CDF/CCDF mappings of z to p/beta and of p/beta to z*

- void `update_final_statistics` ()
  *update finalStatistics from minValues/maxValues, momentStats, and computedProbLevels/computedRelLevels/computedRespLevels*

- void `print_pdf_mappings` (std::ostream &s) const
  *prints the PDFs computed in compute_statistics()*

- void `sampling_reference` (int samples_ref)
  *set samplesRef*

**Protected Member Functions**

- `NonDSampling` (Model &model)<br>
  *constructor*

- `NonDSampling` (NoDBBaseConstructor, Model &model, const String &sample_type, int samples, int seed, const String &rng, bool vary_pattern, short sampling_vars_mode)<br>
  *alternate constructor for sample generation and evaluation "on the fly"*

- `NonDSampling` (NoDBBaseConstructor, const String &sample_type, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)<br>
  *alternate constructor for sample generation "on the fly"*

- `~NonDSampling` ()<br>
  *destructor*
• int num_samples () const
  get the current number of samples

• void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  resets number of samples and sampling flags

• const String & sampling_scheme () const
  return sampleType: "lhs" or "random"

• 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 (const RealVector &lower_bnds, const RealVector &upper_bnds)
  Uses lhsDriver to generate a set of uniform samples over lower_bnds/upper_bnds.

• void update_model_from_sample (Model &model, const Real ∗sample_vars)
  Override default update of continuous vars only.

• void initialize_lhs (bool write_message)
  increments numLHSRuns, sets random seed, and initializes lhsDriver

• 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 (const IntResponseMap &samples)
  called by compute_statistics() to calculate min/max intervals

• void compute_moments (const IntResponseMap &samples)
  called by compute_statistics() to calculate means, std deviations, and confidence intervals

• 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()

• void print_moments (std::ostream &s) const
  prints the moments computed in compute_moments()

• void view_design_counts (const Model &model, size_t &num_cdv, size_t &num_didv, size_t &num_drdv) 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_cauv, size_t &num_diauv, size_t &num_drauv) 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_dieuv, size_t &num_dreuv) 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_diuv, size_t &num_druv) 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_disv, size_t &num_drsv) 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 &drv_start, size_t &num_drv) const
  
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

**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

- String `sampleType`
  - the sample type: random, lhs, or incremental_lhs

- 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

### Private Member Functions

- void `archive_allocate_pdf()`
  - allocate results array storage for pdf histograms

- void `archive_pdf(size_t fn_index)`
  - archive a single pdf histogram for specified function

### Private Attributes

- size_t `numLHSRuns`
  - counter for number of executions of `get_parameter_sets()` for this object
Class Documentation

- **RealMatrix** momentCIs
  
  Matrix of confidence internals on moments, with rows for mean_lower, mean_upper, sd_lower, sd_upper (calculated in compute_moments()).

- **RealMatrix** extremeValues
  
  Minimum (row 0) and maximum (row 1) values of response functions for epistemic calculations (calculated in compute_intervals()).

- **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)

13.101.1 Detailed Description

Base class for common code between NonDLHSSampling, NonDIncremLHSSampling, and NonDAdaptImpSampling. 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 2001. The 1970’s vintage LHS (that had been f2c’d and converted to incomplete classes) has been removed.

13.101.2 Constructor & Destructor Documentation

13.101.2.1 NonDSampling (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.

References Dakota::abort_handler(), NonD::epistemicStats, NonD::initialize_final_statistics(), Iterator::maxConcurrency, NonDSampling::numSamples, NonDSampling::sampleType, and NonD::totalLevelRequests.

13.101.2.2 NonDSampling (NoDBBaseConstructor, Model & model, const String & 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::maxConcurrency, NonD::numEpistemicUncVars, NonDSampling::numSamples, NonDSampling::sampleType, NonDSampling::samplingVarsMode, and Iterator::subIteratorFlag.
13.101.2.3 NonDSampling (NoDBBaseConstructor, const String & 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::maxConcurrency, NonDSampling::numSamples, NonDSampling::sampleType, and Iterator::subIteratorFlag.

13.101.3 Member Function Documentation

13.101.3.1 int num_samples () const [inline, protected, virtual]

get the current number of samples 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 maxConcurrency. May be (is) overridden by derived classes.

Reimplemented from Iterator.

References NonDSampling::numSamples.

Referenced by NonDAdaptImpSampling::generate_samples(), NonDAdaptImpSampling::select_init_rep_points(), and NonDAdaptImpSampling::select_rep_points().

13.101.3.2 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::samplesRef, and NonDSampling::statsFlag.

13.101.3.3 void get_parameter_sets (Model & model) [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.

Reimplemented from Analyzer.

References Dakota::abort_handler(), Model::acv(), Model::adiv(), Model::aleatory_distribution_parameters(), Model::all_continuous_lower_bounds(), Model::all_continuous_upper_bounds(), Model::all_discrete_int_lower_bounds(), Model::all_discrete_int_upper_bounds(), Model::all_discrete_real_lower_bounds(), Model::all_discrete_real_upper_bounds(), Model::all_discrete_set_int_values(), Model::all_discrete_set_int_values(), Model::all_discrete_set_real_values(), Model::all_discrete_set_real_values(), Model::all_discrete_state_set_int_values(), Model::all_discrete_state_set_int_values(), Model::all_discrete_state_set_real_values(), Model::all_discrete_state_set_real_values(), Model::all_aleatory_distribution_parameters(), Model::all_continuous_lower_bounds(), Model::all_continuous_upper_bounds(), Model::all_discrete_int_lower_bounds(), Model::all_discrete_int_upper_bounds(), Model::all_discrete_real_lower_bounds(), Model::all_discrete_real_upper_bounds(), Model::all_discrete_set_int_values(), Model::all_discrete_set_int_values(), Model::all_discrete_set_real_values(), Model::all_discrete_set_real_values(), Model::all_aleatory_distribution_parameters().
Class Documentation

parameters(), NonDSampling::initialize_lhs(), NonDSampling::lhsDriver, NonDSampling::mode_counts(),
NonD::numContDesVars, NonD::numContStateVars, NonD::numDiscIntDesVars, NonD::numDiscIntStateVars,
NonD::numDiscRealDesVars, NonD::numDiscRealStateVars, NonDSampling::numSamples, NonDSampling::sampleRanks,
NonDSampling::samplingVarsMode, and Variables::view().

Referenced by NonDLHSSampling::NonDLHSSampling(), NonDLHSSampling::pre_run(),
NonDIncremLHSSampling::quantify_uncertainty(), and NonDAdaptImpSampling::quantify_uncertainty().

13.101.3.4 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.

13.101.3.5 void view_design_counts (const Model & model, size_t & num_cdv, size_t & num_didv, size_t & num_drdv) 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::all_continuous_variable_types(), Model::all_discrete_int_variable_types(), Model::all_discrete_real_variable_types(), Model::current_variables(), Variables::cv_start(), Variables::div_start(), Variables::drv_start(), NonD::numContDesVars, NonD::numDiscIntDesVars, NonD::numDiscRealDesVars, and Variables::view().

Referenced by NonDSampling::mode_counts().

13.101.3.6 void view_aleatory_uncertain_counts (const Model & model, size_t & num_cauv, size_t & num_diauv, size_t & num_drauv) 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 Model::aleatory_distribution_parameters(), Model::current_variables(), Variables::cv(), Variables::div(), Variables::drv(), NonD::numContAleatUncVars, NonD::numDiscIntAleatUncVars, NonD::numDiscRealAleatUncVars, and Variables::view().

Referenced by NonDSampling::mode_counts().
13.101.3.7  void view_epistemic_uncertain_counts (const Model & model, size_t & num_ceuv, size_t & num_dieuv, size_t & num_dreuv) 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(), Variables::cv(), Variables::div(), Variables::drv(), Model::epistemic_distribution_parameters(), NonD::numContEpistUncVars, NonD::numDiscIntEpistUncVars, NonD::numDiscRealEpistUncVars, and Variables::view().

Referenced by NonDSampling::mode_counts().

13.101.3.8  void view_uncertain_counts (const Model & model, size_t & num_cuv, size_t & num_diuv, size_t & num_druv) 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::aleatory_distribution_parameters(), Model::current_variables(), Variables::cv(), Variables::div(), Variables::drv(), Model::epistemic_distribution_parameters(), NonD::numContAleatUncVars, NonD::numContEpistUncVars, NonD::numDiscIntAleatUncVars, NonD::numDiscIntEpistUncVars, NonD::numDiscRealAleatUncVars, NonD::numDiscRealEpistUncVars, and Variables::view().

Referenced by NonDSampling::mode_counts().

The documentation for this class was generated from the following files:

- NonDSampling.hpp
- NonDSampling.cpp
13.102 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:

```
NonDSparseGrid
   NonDIntegration
   NonD
   Analyzer
   Iterator
```

Public Member Functions

- **NonDSparseGrid (Model &model, short exp_coeffs_approach, const UShortArray &ssg_level_seq, const RealVector &dim_pref, 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 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 increment_specification_sequence ()**
  
  `advance to next nevel in ssgLevelSeqSpec sequence`

- **const std::set<UShortArray> & active_multi_index () const**
  
  `returns SparseGridDriver::active_multi_index()`

- **const std::set<UShortArray> & old_multi_index () const**
  
  `returns SparseGridDriver::old_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()

• void update_sets (const UShortArray &set_star)
  invokes SparseGridDriver::update_sets()

• void print_final_sets (bool converged_within_tol)
  invokes SparseGridDriver::print_final_sets(bool)

• void finalize_sets ()
  invokes SparseGridDriver::finalize_sets()

• void evaluate_grid_increment ()
  invokes SparseGridDriver::evaluate_grid_increment()

• int num_samples () const
  get the current number of samples

Protected Member Functions

• NonDSparseGrid (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)
  Returns one block of samples (ndim * num_samples).
• void reset ()
  restore initial state for repeated sub-iterator executions

• void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)

### Private Attributes

- Pecos::SparseGridDriver * ssgDriver
  convenience pointer to the numIntDriver representation

- UShortArray ssgLevelSeqSpec
  the user specification for the Smolyak sparse grid level, defining a sequence of refinement levels.

- unsigned short ssgLevelRef
  reference point (e.g., lower bound) for the Smolyak sparse grid level maintained within ssgDriver

### 13.102.1 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.

### 13.102.2 Constructor & Destructor Documentation

13.102.2.1 NonDSparseGrid (Model & model, short exp_coeffs_approach, const UShortArray & ssg_level_seq, const RealVector & dim_pref, 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.

13.102.2.2 NonDSparseGrid (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(), NonDIntegration::dimPrefSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_short(), Iterator::iteratedModel, Iterator::maxConcurrency, NonD::natafTransform, NonDIntegration::numIntDriver, Iterator::probDescDB, NonDSparseGrid::ssgDriver, and NonDSparseGrid::ssgLevelRef.
13.102.3 Member Function Documentation

13.102.3.1 int num_samples () const [inline, virtual]

get the current number of samples. 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 maxConcurrency. May be (is) overridden by derived classes.

Reimplemented from Iterator.

References NonDSparseGrid::ssgDriver.

13.102.3.2 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
13.103 NonDStochCollocation Class Reference

Nonintrusive stochastic collocation approaches to uncertainty quantification. Inheritance diagram for NonD-StochCollocation:

Public Member Functions

- **NonDStochCollocation (Model &model)**
  
  *standard constructor*

- **NonDStochCollocation (Model &model, short exp_coeffs_approach, unsigned short num_int_level, short u_space_type, bool piecewise_basis, bool use_derivs)**
  
  *alternate constructor*

- **~NonDStochCollocation ()**
  
  *destructor*

Protected Member Functions

- **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 ()**
  
  *update an expansion; avoids overhead in compute_expansion()*

- **Real compute_covariance_metric ()**
  
  *compute 2-norm of change in response covariance*
13.103 NonDStochCollocation Class Reference

- Real compute_final_statistics_metric()
  
  compute 2-norm of change in final statistics

Private Attributes

- short sgBasisType
  
  Type of interpolant (from enum in DataMethod.hpp).

13.103.1 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.

13.103.2 Constructor & Destructor Documentation

13.103.2.1 NonDStochCollocation (Model & model)

standard constructor This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Model::assign_rep(), NonDExpansion::construct_expansion_sampler(), NonDExpansion::construct_ -quadrature(), NonDExpansion::construct_sparse_grid(), Model::derivative_concurrency(), NonDExpansion::expansionCoeffsApproach, ProblemDescDB::get_bool(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_usa(), Model::init_, communicator(), NonDExpansion::initialize(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, NonDExpansion::nestedRules, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, NonDExpansion::numSamplesOnExpansion, Iterator::outputLevel, NonDExpansion::piecewiseBasis, Iterator::probDescDB, NonDExpansion::refineControl, NonDStochCollocation::resolve_, inputs(), NonDStochCollocation::sgBasisType, NonD::transform_model(), and NonDExpansion::uSpaceModel.

13.103.2.2 NonDStochCollocation (Model & model, short exp_coeffs_approach, unsigned short num_int_level, short u_space_type, bool piecewise_basis, bool use_derivs)

alternate constructor This constructor is used for helper iterator instantiation on the fly.

References Model::assign_rep(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_- sparse_grid(), NonDExpansion::expansionCoeffsApproach, NonDExpansion::initialize(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Iterator::outputLevel, NonDExpansion::piecewiseBasis, NonDStochCollocation::resolve_inputs(), NonDStochCollocation::sgBasisType, NonD::transform_model(), and NonDExpansion::uSpaceModel.
13.103.3 Member Function Documentation

13.103.3.1 Real compute_c 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(), PecosApproximation::delta_covariance(), PecosApproximation::expansion_coefficient_flag(), NonDExpansion::initialPtU, NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Iterator::numFunctions, NonDExpansion::respCovariance, NonDStochCollocation::sgBasisType, and NonDExpansion::uSpaceModel.

13.103.3.2 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_statistics(), PecosApproximation::delta_beta(), PecosApproximation::delta_z(), PecosApproximation::expansion_coefficient_flag(), NonD::finalStatistics, Response::function_values(), NonDExpansion::initialPtU, Response::num_functions(), NonD::numContDesVars, NonD::numContEpistUncVars, NonD::numContStateVars, Iterator::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, NonDStochCollocation::sgBasisType, NonDExpansion::uSpaceModel, and Dakota::write_data().

The documentation for this class was generated from the following files:

- NonDStochCollocation.hpp
- NonDStochCollocation.cpp
13.104 NonlinearCGOptimizer Class Reference

Inheritance diagram for NonlinearCGOptimizer::

```
Iterator
    Minimizer
        Optimizer
            NonlinearCGOptimizer
```

Public Member Functions

- **NonlinearCGOptimizer (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 find_optimum ()**
  
  *Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.*

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 abissca 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
13.104 NonlinearCGOptimizer Class Reference

... (details omitted for brevity) ...

13.104.1 Detailed Description

Experimental implementation of nonlinear CG optimization

13.104.2 Member Function Documentation

13.104.2.1 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
13.105  NPSOLOptimizer Class Reference

Wrapper class for the NPSOL optimization library. Inheritance diagram for NPSOLOptimizer::

```
  Iterator
   |
   Minimizer
   |
  Optimizer  SOLBase
  |        |
  NPSOLOptimizer
```

**Public Member Functions**

- **NPSOLOptimizer** (Model &model)
  
  *standard constructor*

- **NPSOLOptimizer** (NoDBBaseConstructor, Model &model)
  
  *alternate constructor for Iterator instantiations by name*

- **NPSOLOptimizer** (Model &model, const int &derivative_level, const Real &conv_tol)
  
  *alternate constructor for instantiations "on the fly"*

- **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 *, int &), void(*user_con_eval)(int &, int &, int &, int &, int *, double *, double *, double *, int &), const int &derivative_level, const Real &conv_tol)
  
  *alternate constructor for instantiations "on the fly"*

- **~NPSOLOptimizer**()
  
  *destructor*

- **void find_optimum**()
  
  *Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.*

**Private Member Functions**

- **void find_optimum_on_model**()
called by find_optimum for setUpType == "model"

- void find_optimum_on_user_functions ()
  called by find_optimum for setUpType == "user_functions"

Static Private Member Functions

- static void objective_eval (int &mode, int &n, double *x, double &f, double *gradf, int &nstate)
  OBJFUN in NPSOL manual: computes the value and first derivatives of the objective function (passed by function pointer to NPSOL).

Private Attributes

- String setUpType
  controls iteration mode: "model" (normal usage) or "user_functions" (user-supplied functions mode for "on the fly" instantiations). NonDReliability currently uses the user_functions mode.

- RealVector initialPoint
  holds initial point passed in for "user_functions" mode.

- RealVector lowerBounds
  holds variable lower bounds passed in for "user_functions" mode.

- RealVector upperBounds
  holds variable upper bounds passed in for "user_functions" mode.

- void(* userObjectiveEval )(int &, int &, double *, double &, double *, double *, int &)
  holds function pointer for objective function evaluator passed in for "user_functions" mode.

- void(* userConstraintEval )(int &, int &, int &, int &, int *, int *, double *, double *, double *, double *, int &)
  holds function pointer for constraint function evaluator passed in for "user_functions" mode.

Static Private Attributes

- static NPSOLOptimizer * npsolInstance
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
13.105 NPSOLOptimizer Class Reference

13.105.1 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.

13.105.2 Constructor & Destructor Documentation

13.105.2.1 NPSOLOptimizer (Model & model)

standard constructor This is the primary constructor. It accepts a Model reference.

References Minimizer::constraintTol, Iterator::convergenceTol, Iterator::fdGradStepSize, ProblemDescDB::get_int(), ProblemDescDB::get_real(), Iterator::gradientType, Iterator::maxIterations, Iterator::outputLevel, Iterator::probDescDB, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

13.105.2.2 NPSOLOptimizer (NoDBBaseConstructor, Model & model)

alternate constructor for Iterator instantiations by name This is an alternate constructor which accepts a Model but does not have a supporting method specification from the ProblemDescDB.

References Minimizer::constraintTol, Iterator::convergenceTol, Iterator::fdGradStepSize, Iterator::gradientType, Iterator::maxIterations, Iterator::outputLevel, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

13.105.2.3 NPSOLOptimizer (Model & model, const int & derivative_level, const Real & conv_tol)

alternate constructor for instantiations "on the fly" This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.
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 & lin_eq_lower_bnds, const RealVector & lin_eq_upper_bnds, const RealVector & nonlin_ineq_lower_bnds, const RealVector & nonlin_ineq_upper_bnds, const RealVector & nonlin_eq_targets, void(*)(int &, int &, double *, double &, double *, int &) user_obj_eval, void(*)(int &, int &, int &, int &, int *, double *, double *, double *, int &) user_con_eval, const int & derivative_level, const Real & conv_tol)

alternate constructor for instantiations "on the fly" This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers.

References SOLBase::allocate_arrays(), SOLBase::allocate_workspace(), SOLBase::augment_bounds(), NPSOLOptimizer::lowerBounds, Iterator::numContinuousVars, Minimizer::numLinearConstraints, Minimizer::numNonlinearConstraints, and NPSOLOptimizer::upperBounds.

The documentation for this class was generated from the following files:

- NPSOLOptimizer.hpp
- NPSOLOptimizer.cpp
13.106 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy. Inheritance diagram for Optimizer::

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 ()
  
  default constructor

- Optimizer (Model &model)
  
  standard constructor
• **Optimizer (NoDBBaseConstructor, Model &model)**
  alternate constructor for "on the fly" instantiations

• **Optimizer (NoDBBaseConstructor, size_t num_cv, size_t num_div, size_t num_drv, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq)**
  alternate constructor for "on the fly" instantiations

• **∼Optimizer ()**
  destructor

• void **initialize_run ()**
• void **run ()**
  run portion of run_iterator; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

• 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)**
• virtual void **find_optimum ()=0**
  Used within the optimizer branch for computing the optimal solution. Redefines the run virtual function for the optimizer branch.

### 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 optimizerInstance

### 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

- void **local_objective_recast_retrieve** (const Variables &vars, Response &response) const
  
  infers MOO/NLS solution from the solution of a single-objective optimizer

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.

13.106.1 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, NLPQLPOptimizer, COLINOptimizer, and JEGAOptimizer.

13.106.2 Constructor & Destructor Documentation

13.106.2.1 Optimizer (Model & model) [protected]

standard constructor This constructor extracts the inherited data for the optimizer branch and performs sanity checking on gradient and constraint settings.

References Dakota::abort_handler(), Iterator::bestVariablesArray, Minimizer::boundConstraintFlag, Variables::copy(), Model::current_variables(), Minimizer::data_transform_model(), ProblemDescDB::get_sizet(), Iterator::gradientType, Iterator::hessianType, Model::init_communicators(), Iterator::iteratedModel, Optimizer::localObjectiveRecast, Iterator::maxConcurrency, Iterator::methodName, Minimizer::minimizerRecasts, Model::model_type(), Minimizer::numIterPrimaryFns, Optimizer::numObjectiveFns, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, Minimizer::optimizationFlag, Model::primary_response_fn_weights(), Iterator::probDescDB, Optimizer::reduce_model(), Minimizer::scale_model(), Minimizer::scaleFlag, Minimizer::speculativeFlag, and Dakota::strbegins().
13.106.3 Member Function Documentation

13.106.3.1 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.

Reimplemented in CONMINOptimizer, DOTOptimizer, NLPQLPOptimizer, and SNLLOptimizer.

References Iterator::iteratedModel, Minimizer::minimizerRecasts, Optimizer::optimizerInstance, Optimizer::prevOptInstance, and Model::update_from_subordinate_model().

13.106.3.2 void run () [inline, protected, virtual]

run portion of run_iterator; 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 Optimizer::find_optimum().

13.106.3.3 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 Iterator.

Reimplemented in COLINOptimizer, and SNLLOptimizer.

References Dakota::abort_handler(), Response::active_set_request_vector(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Variables::continuous_variables(), Response::copy(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleOffsets, Minimizer::cvScaleTypes, Minimizer::expData, Response::function_value(), Response::function_values(), Optimizer::localObjectiveRecast, Minimizer::modify_s2n(), Minimizer::need_resp_trans_byvars(), Minimizer::numExperiments, Minimizer::numNonlinearConstraints, Minimizer::numReplicates, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, Minimizer::primaryRespScaleFlag, Minimizer::response_modify_s2n(), Minimizer::secondaryRespScaleFlag, Response::update_partial(), and Minimizer::varsScaleFlag.

13.106.3.4 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.

Reimplemented in SNLLOptimizer.

References Optimizer::optimizerInstance, and Optimizer::prevOptInstance.
13.106.3.5 void print_results (std::ostream & s) [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, Dakota::data_pairs, Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Iterator::numContinuousVars, Iterator::numFunctions, Minimizer::numNonlinearConstraints, Minimizer::optimizationFlag, Model::primary_response_fn_weights(), Model::subordinate_model(), Dakota::write_data_partial(), and Dakota::write_precision.

13.106.3.6 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(), Model::current_response(), Minimizer::gnewton_set_recast(), Iterator::hessianType, Iterator::iteratedModel, Iterator::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Optimizer::numObjectiveFns, Minimizer::numRowsExpData, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, Iterator::outputLevel, Optimizer::primary_resp_reducer(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), ActiveSet::request_vector(), Response::reshape(), and Minimizer::secondary_resp_copier().

Referenced by Optimizer::Optimizer().

13.106.3.7 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().

13.106.3.8 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, Dakota::write_column_vector_trans(), Dakota::write_data(), and Dakota::write_precision.

Referenced by Optimizer::primary_resp_reducer().

13.106.3.9 void local_objective_recast_retrieve (const Variables & vars, Response & response) const

[private]

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.

References Response::active_set(), Dakota::data_pairs, Response::function_value(), Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Minimizer::numRowsExpData, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, and Response::update().

Referenced by Optimizer::post_run().

The documentation for this class was generated from the following files:

- DakotaOptimizer.hpp
- DakotaOptimizer.cpp
13.107 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 wPLIter

- **const ParallelLevel & si_parallel_level () const**
  
  return the ParallelLevel corresponding to siPLIter

- **const ParallelLevel & ie_parallel_level () const**
  
  return the ParallelLevel corresponding to iePLIter

- **const ParallelLevel & ea_parallel_level () const**
  
  return the ParallelLevel corresponding to eaPLIter

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

- **ParLevLIter wPLIter**
list iterator for MPI_COMM_WORLD (not strictly required, but improves modularity by avoiding explicit usage of MPI_COMM_WORLD)

- ParLevLIter siPLIter
  list iterator for concurrent iterator partitions (there may be more than one 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)

Friends

- class ParallelLibrary
  the ParallelLibrary class has special access privileges in order to streamline implementation

13.107.1 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).

The documentation for this class was generated from the following file:

- ParallelLibrary.hpp
13.108  ParallelDirectApplicInterface Class Reference

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

```
Class Hierarchy:
  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_iterator_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*

13.108.1 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 may be activated by specifying the --with-plugin configure option, which activates the DAKOTA_PLUGIN macro in dakota_config.h used by main.cpp (which activates the plug-in code block within that file) and activates the PLUGIN_S declaration defined in Makefile.include and used in Makefile.source (which add this class to the build). Test input files should then use an analysis_driver of "plugin_textbook".

13.108.2 Member Function Documentation

13.108.2.1 void test_local_evaluations (Dakota::PRPQueue & prp_queue) [inline, protected, virtual]

invokes wait_local_evaluations() (no special nowait support) For use by ApplicationInterface::serve_evaluations_async(), which can provide a batch processing capability within message passing schedulers (called using chain ApplicationInterface::serve_evaluations() from Model::serve() from Strategy::run_iterator()).

Reimplemented from DirectApplicInterface.

References ParallelDirectApplicInterface::wait_local_evaluations().

The documentation for this class was generated from the following files:

- PluginParallelDirectApplicInterface.hpp
- PluginParallelDirectApplicInterface.cpp
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_flag () const`
  
  return dedicatedMasterFlag

- `bool communicator_split_flag () const`
  
  return commSplitFlag

- `bool server_master_flag () const`
  
  return serverMasterFlag

- `bool message_pass () const`
  
  return messagePass

- `const int & num_servers () const`
  
  return numServers

- `const int & processors_per_server () const`
  
  return procsPerServer

- `const int & processor_remainder () const`
  
  return procRemainder

- `const MPI_Comm & server_intra_communicator () const`
  
  return serverIntraComm

- `const int & server_communicator_rank () const`
  
  return serverCommRank
• const int & server_communicator_size () const
  return serverCommSize

• const MPI_Comm & hub_server_intra_communicator () const
  return hubServerIntraComm

• const int & hub_server_communicator_rank () const
  return hubServerCommRank

• const 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

• const int & server_id () const
  return serverId

Private Member Functions

• void assign (const ParallelLevel &pl)
  assign the attributes of the incoming pl to this object

Private Attributes

• 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

• int numServers
  number of servers
• int procsPerServer
  processors per server

• int procRemainder
  proc remainder after equal distribution

• 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

• int serverId
  server identifier

Friends
• class ParallelLibrary
  the ParallelLibrary class has special access privileges in order to streamline implementation

13.109.1 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.

The documentation for this class was generated from the following file:
- ParallelLibrary.hpp
13.110 ParallelLibrary Class Reference

Class for partitioning multiple levels of parallelism and managing message passing within these levels.

Public Member Functions

- **ParallelLibrary** (int &argc, char **&argv)
  stand-alone mode constructor

- **ParallelLibrary()**
  default library mode constructor (assumes MPI_COMM_WORLD)

- **ParallelLibrary** (MPI_Comm dakota_mpi_comm)
  library mode constructor accepting communicator

- **ParallelLibrary** (const std::string &dummy)
  dummy constructor (used for dummy_lib)

- **~ParallelLibrary()**
  destructor

- **const ParallelLevel & init_iterator_communicators** (const int &iterator_servers, const int &procs_per_iterator, const int &max_iterator_concurrency, const std::string &default_config, const std::string &iterator_scheduling)
  split MPI_COMM_WORLD into iterator communicators

- **const ParallelLevel & init_evaluation_communicators** (const int &evaluation_servers, const int &procs_per_evaluation, const int &max_evaluation_concurrency, const int &asynch_local_evaluation_concurrency, const std::string &default_config, const std::string &evaluation_scheduling)
  split an iterator communicator into evaluation communicators

- **const ParallelLevel & init_analysis_communicators** (const int &analysis_servers, const int &procs_per_analysis, const int &max_analysis_concurrency, const int &asynch_local_analysis_concurrency, const std::string &default_config, const std::string &analysis_scheduling)
  split an evaluation communicator into analysis communicators

- **void free_iterator_communicators()**
  deallocate iterator communicators

- **void free_evaluation_communicators()**
  deallocate evaluation communicators

- **void free_analysis_communicators()**
  deallocate analysis communicators

- **void print_configuration()**
print the parallel level settings for a particular parallel configuration

- void `specify_outputs_restart` (CommandLineHandler &cmd_line_handler)
  specify output streams and restart file(s) using command line inputs (normal mode)

- void `specify_outputs_restart` (const char *clh_std_output_filename=NULL, const char *clh_std_error_filename=NULL, const char *clh_read_restart_filename=NULL, const char *clh_write_restart_filename=NULL, int stop_restart_evals=0, bool pre_run_flag=false)
  specify output streams and restart file(s) using external inputs (library mode).

- void `manage_outputs_restart` (const ParallelLevel &pl, bool results_output=false, std::string results_filename)
  manage output streams and restart file(s) (both modes)

- void `close_streams`()
  close streams, files, and any other services

- void `abort_helper` (int code) const
  finalize MPI with correct communicator for abort

- const std::string & `startup_message` () const
  get pretty startup message

- void `output_helper` (const std::string &message, std::ostream &os=Cout) const
  perform output of message on rank 0 only

- 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 std::string & `command_line_pre_run_input` () const
  preRunInput filename

- const std::string & `command_line_pre_run_output` () const
  preRunOutput filename
• const std::string & command_line_run_input () const
  runInput filename

• const std::string & command_line_run_output () const
  runOutput filename

• const std::string & command_line_post_run_input () const
  postRunInput filename

• const std::string & command_line_post_run_output () const
  postRunOutput fname

• void send_si (int &send_int, int dest, int tag)
  blocking send at the strategy-iterator communication level

• void recv_si (int &recv_int, int source, int tag, MPI_Status &status)
  blocking receive at the strategy-iterator communication level

• void send_si (MPIPackBuffer &send_buff, int dest, int tag)
  blocking send at the strategy-iterator communication level

• void isend_si (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req)
  nonblocking send at the strategy-iterator communication level

• void recv_si (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status)
  blocking receive at the strategy-iterator communication level

• void irecv_si (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req)
  nonblocking receive at the strategy-iterator 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_w (int &data)
  broadcast an integer across MPI_COMM_WORLD

• void bcast_i (int &data)
  broadcast an integer across an iterator communicator

• void bcast_i (short &data)
  broadcast a short integer across an iterator communicator

• void bcast_e (int &data)
  broadcast an integer across an evaluation communicator

• void bcast_a (int &data)
  broadcast an integer across an analysis communicator

• void bcast_si (int &data)
  broadcast an integer across a strategy-iterator intra communicator

• void bcast_w (MPIPackBuffer &send_buff)
  broadcast a packed buffer across MPI_COMM_WORLD

• void bcast_i (MPIPackBuffer &send_buff)
  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_si (MPIPackBuffer &send_buff)
  broadcast a packed buffer across a strategy-iterator intra communicator

• void bcast_w (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer broadcast across MPI_COMM_WORLD

• void bcast_i (MPIUnpackBuffer &recv_buff)
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_si (MPIUnpackBuffer &recv_buff)
  matching recv for packed buffer bcast across a strat-iterator intra comm

• void barrier_w ()
  enforce MPI_Barrier on MPI_COMM_WORLD

• void barrier_i ()
  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, const 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, const 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 (const int &num_recvs, MPI_Request *&recv_reqs)
  wait for all messages from a series of nonblocking receives

• void waitsome (const int &num_sends, MPI_Request *&recv_requests, int &num_recvs, 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

• const int & world_size () const
return worldSize

- const int & world_rank () const
  return worldRank

- bool mpirun_flag () const
  return mpirunFlag

- bool is_null () const
  return dummyFlag

- Real parallel_time () const
  returns current MPI wall clock time

- void parallel_configuration_iterator (const ParConfigLIter &pc_iter)
  set the current ParallelConfiguration node

- const 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 ()
  add a new node to parallelConfigurations and increment currPCIter

- bool w_parallel_level_defined () const
  test current parallel configuration for definition of world parallel level

- bool si_parallel_level_defined () const
  test current parallel configuration for definition of strategy-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

- std::vector<MPI_Comm> 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).
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 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, const int &num_servers, const int &procs_per_server, const int &max_concurrency, const int &asynch_local_concurrency, const std::string &default_config, const std::string &scheduling_override)
  split a parent communicator into child server communicators

- void free_communicators (ParallelLevel &pl)
  deallocate intra/inter communicators for a particular ParallelLevel

- bool 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

- bool 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)

- bool resolve_inputs (int &num_servers, int &procs_per_server, const int &avail_procs, int &proc_remainder, const int &max_concurrency, const int &capacity_multiplier, const std::string &default_config, const std::string &scheduling_override, bool print_rank)
  resolve user inputs into a sensible partitioning scheme

- void send (MPIPackBuffer &send_buff, const int &dest, const int &tag, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking buffer send at the current communication level

- void send (int &send_int, const int &dest, const int &tag, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking integer send at the current communication level

- void isend (MPIPackBuffer &send_buff, const int &dest, const int &tag, MPI_Request &send_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking buffer send at the current communication level
nonblocking buffer send at the current communication level

- void isend (int &send_int, const int &dest, const int &tag, MPI_Request &send_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking integer send at the current communication level

- void recv (MPIUnpackBuffer &recv_buff, const int &source, const int &tag, MPI_Status &status, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking buffer receive at the current communication level

- void recv (int &recv_int, const int &source, const int &tag, MPI_Status &status, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking integer receive at the current communication level

- void irecv (MPIUnpackBuffer &recv_buff, const int &source, const int &tag, MPI_Request &recv_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking buffer receive at the current communication level

- void irecv (int &recv_int, const int &source, const int &tag, MPI_Request &recv_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking integer receive at the current communication level

- 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

- 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, const int &num_vals, const MPI_Comm &comm)
  compute a sum over comm using MPI_Reduce

- void check_error (const std::string &err_source, const int &err_code)
  check the MPI return code and abort if error

- void manage_run_modes (CommandLineHandler &cmd_line_handler)
  manage run mode information from command-line handler
void split_filenames (const char *filenames, std::string &input_filename, std::string &output_filename)
    split a double colon separated pair of filenames (possibly empty) into input and output filename strings

void assign_streams (bool append)
    potentially reassign streams during initial specification of output and error files; may later get overridden with
tagged files.

Private Attributes

std::ofstream output_ofstream
    tagged file redirection of stdout

std::ofstream error_ofstream
    tagged file redirection of stderr

MPI_Comm dakotaMPIComm
    MPI_Comm on which DAKOTA is running.

int worldRank
    rank in MPI_Comm in which DAKOTA is running

int worldSize
    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

bool dummyFlag
    prevents multiple MPI_Finalize calls due to dummy_lib

bool stdOutputToFile
    flags redirection of DAKOTA std output to a file

bool stdErrorToFile
    flags redirection of DAKOTA std error to a file

std::string startupMessage
    cached startup message for use in check_inputs

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 user run modes are active

• bool outputTimings
  timing info only beyond help/version/check

• std::string preRunInput
  filename for pre_run input

• std::string preRunOutput
  filename for pre_run output

• std::string runInput
  filename for run input

• std::string runOutput
  filename for run output

• std::string postRunInput
  filename for post_run input

• std::string postRunOutput
  filename for post_run output

• Real startCPUTime
  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::string stdOutputFilename
13.110 ParallelLibrary Class Reference

```cpp

filename for redirection of stdout

- std::string stdErrorFilename
  filename for redirection of stderr

- std::string readRestartFilename
  input filename for restart

- std::string writeRestartFilename
  output filename for restart

- int stopRestartEvals
  number of evals at which to stop restart processing

- 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

- ParLevLIter currPLIter
  list iterator identifying the current node in parallelLevels

- ParConfigLIter currPCIter
  list iterator identifying the current node in parallelConfigurations
```

### 13.110.1 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.

### 13.110.2 Constructor & Destructor Documentation

#### 13.110.2.1 ParallelLibrary (int & argc, char **& argv)

Stand-alone mode constructor This constructor is the one used by `main.cpp`. It calls MPI_Init conditionally based on whether a parallel launch is detected.

References

- ParallelLibrary::detect_parallel_launch()
- ParallelLibrary::init_mpi_comm()
- ParallelLibrary::initialize_timers()
- ParallelLibrary::mpirunFlag, and ParallelLibrary::ownMPIFlag.

Dakota Version 5.4 Developers Manual generated on November 7, 2013
13.110.2.2 ParallelLibrary()

default library mode constructor (assumes MPI_COMM_WORLD) This constructor provides a library mode default ParallelLibrary. It does not call MPI_Init, but rather gathers data from MPI_COMM_WORLD if MPI_Init has been called elsewhere.
References ParallelLibrary::init_mpi_comm(), ParallelLibrary::initialize_timers(), and ParallelLibrary::mpirunFlag.

13.110.2.3 ParallelLibrary(MPI_Comm dakota_mpi_comm)

library mode constructor accepting communicator This constructor provides a library mode ParallelLibrary, accepting an MPI communicator that might not be MPI_COMM_WORLD. It does not call MPI_Init, but rather gathers data from dakota_mpi_comm if MPI_Init has been called elsewhere.
References ParallelLibrary::init_mpi_comm(), ParallelLibrary::initialize_timers(), and ParallelLibrary::mpirunFlag.

13.110.2.4 ParallelLibrary(const std::string & dummy)
dummy 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.

13.110.3 Member Function Documentation

13.110.3.1 void specify_outputs_restart(CommandLineHandler & cmd_line_handler)
specify output streams and restart file(s) using command line inputs (normal mode) On the rank 0 processor, get the -output, -error, -read_restart, and -write_restart filenames and the -stop_restart limit from the command line. Defaults for the filenames from the command line handler are NULL for the filenames except write which defaults to dakota.rst and 0 for read_restart_evals if no user specification. This information is Bcast from rank 0 to all iterator masters in manage_outputs_restart().
References ParallelLibrary::assign_streams(), ParallelLibrary::manage_run_modes(), CommandLineHandler::read_restart_evals(), ParallelLibrary::readRestartFilename, GetLongOpt::retrieve(), ParallelLibrary::stdErrorFilename, ParallelLibrary::stdOutputFilename, ParallelLibrary::stopRestartEvals, ParallelLibrary::worldRank, and ParallelLibrary::writeRestartFilename.
Referenced by main(), and run_dakota().

13.110.3.2 void specify_outputs_restart(const char ∗ clh_std_output_filename = NULL, const char ∗ clh_std_error_filename = NULL, const char ∗ clh_read_restart_filename = NULL, const char ∗ clh_write_restart_filename = NULL, int stop_restart_evals = 0, bool pre_run_flag = false)
specify output streams and restart file(s) using external inputs (library mode). Rather than extracting from the command line, pass the std output, std error, read restart, and write restart filenames and the stop restart limit directly. This function only needs to be invoked to specify non-default values [defaults for the filenames are NULL (resulting in no output redirection, no restart read, and default restart write) and 0 for the stop restart limit (resulting in no restart read limit)].
13.110.3.3  void manage_outputs_restart (const ParallelLevel & pl, bool results_output = false, std::string results_filename = std::string())

manage output streams and restart file(s) (both modes) 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 Dakota::abort_handler(), ParallelLibrary::bcast(), ParallelLibrary::checkFlag, Dakota::dakota_cerr, Dakota::dakota_cout, Dakota::data_pairs, ParallelLevel::dedicatedMasterFlag, ParallelLibrary::error_ofstream, ParamResponsePair::eval_id(), ParallelLevel::hubServerCommSize, ParallelLevel::hubServerIntraComm, ResultsManager::initialize(), Dakota::iterator_results_db, ParallelLevel::numServers, ParallelLibrary::output_ofstream, ParallelLibrary::postRunFlag, ParallelLibrary::_postRunInput, ParallelLibrary::postRunOutput, ParallelLibrary::preRunFlag, ParallelLibrary::preRunInput, ParallelLibrary::preRunOutput, ParallelLibrary::readRestartFilename, ParallelLibrary::runFlag, ParallelLibrary::runInput, ParallelLibrary::runOutput, ParallelLevel::serverCommRank, ParallelLevel::serverId, ParallelLevel::serverMasterFlag, MPIPackBuffer::size(), ParallelLibrary::stdErrorFilename, ParallelLibrary::stdErrorToFile, ParallelLibrary::stdOutputToFile, and Dakota::write_restart.

Referenced by Strategy::init_iterator_parallelism().

13.110.3.4  void close_streams ()

close streams, files, and any other services Close streams associated with manage_outputs and manage_restart and terminate any additional services that may be active.

References Dakota::abort_handler(), ParallelLibrary::currPCIter, Dakota::dakota_cerr, Dakota::dakota_cout, Dakota::dc_ptr_int, ParallelLibrary::error_ofstream, Dakota::mc_ptr_int, ParallelLibrary::output_ofstream, ParallelLibrary::parallelLevels, ParallelLevel::serverMasterFlag, ParallelLibrary::stdErrorToFile, ParallelLibrary::stdOutputToFile, and Dakota::write_restart.

Referenced by ParallelLibrary::~ParallelLibrary().

13.110.3.5  void increment_parallel_configuration () [inline]

add a new node to parallelConfigurations and increment currPCIter 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 and strategy-iterator parallel levels from the first partial configuration).

References ParallelLibrary::currPCIter, ParallelConfiguration::eaPLIter, ParallelConfiguration::iePLIter, ParallelConfiguration::numParallelLevels, ParallelLibrary::parallelConfigurations, ParallelLibrary::parallelLevels, ParallelLibrary::stdErrorToFile, ParallelLibrary::stdOutputToFile, and Dakota::write_restart.

Referenced by Dakota::write_restart.
lelConfiguration::siPLIter, ParallelLibrary::worldSize, and ParallelConfiguration::wPLIter.
Referenced by Model::init_communicators(), and ParallelLibrary::init_mpi_comm().

13.110.3.6 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(), ParallelLibrary::currPLIter, Dakota::Dak_pl, ParallelLibrary::dakotaMPIComm, ParallelLibrary::increment_parallel_configuration(), ParallelLibrary::mpirunFlag, ParallelLibrary::parallelLevels, ParallelLevel::serverCommRank, ParallelLevel::serverCommSize, ParallelLevel::serverIntraComm, Dakota::start_dakota_heartbeat(), ParallelLibrary::startMPITime, ParallelLibrary::startupMessage, ParallelLibrary::worldRank, and ParallelLibrary::worldSize.
Referenced by ParallelLibrary::ParallelLibrary().

13.110.3.7 void init_communicators(const ParallelLevel &parent_pl, const int &num_servers, const int &procs_per_server, const int &max_concurrency, const int &asynch_local_concurrency, const std::string &default_config, const std::string &scheduling_override) [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 function is called from the Strategy constructor for the concurrent iterator level and from ApplicationInterface::init_communicators() for the concurrent evaluation and concurrent analysis levels.
References ParallelLevel::commSplitFlag, ParallelLibrary::currPCLIter, ParallelLibrary::currPLIter, ParallelLevel::dedicatedMasterFlag, ParallelLevel::numServers, ParallelLibrary::parallelLevels, ParallelLevel::procRemainder, 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().

13.110.3.8 bool resolve_inputs(int &num_servers, int &procs_per_server, int &proc_remainder, const int &max_concurrency, const int &capacity_multiplier, const std::string &default_config, const std::string &scheduling_override, 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::strbegins().
Referenced by ParallelLibrary::init_communicators().
13.110.3.9  void split_filenames (const char * filenames, std::string & input_filename, std::string & output_filename)  [private]

split a double colon separated pair of filenames (possibly empty) into input and output filename strings Tokenize colon-delimited input and output filenames, returns unchanged strings if tokens not found.

Referenced by ParallelLibrary::manage_run_modes().

The documentation for this class was generated from the following files:

- ParallelLibrary.hpp
- ParallelLibrary.cpp
13.111  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) const**
  
  write a ParamResponsePair object in tabular format to an std::ostream

- **void read (BiStream &s)**
  
  read a ParamResponsePair object from the binary restart stream

- **void write (BoStream &s) const**
write a `ParamResponsePair` object to the binary restart stream

- `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

- `const String & interface_id () const`
  - return the interface identifier from the response object

- `const IntStringPair & eval_interface_ids () const`
  - return the aggregate eval/interface identifier from the response object

- `const Variables & prp_parameters () const`
  - return the parameters object

- `const Response & prp_response () const`
  - return the response object

- `void prp_response (const Response &response)`
  - 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 Attributes**

- `Variables prPairParameters`
  - the set of parameters for the function evaluation

- `Response prPairResponse`
  - the response set for the function evaluation

- `IntStringPair evalInterfaceIds`
  - the evalInterfaceIds aggregate
Friends

- bool operator==(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  equality operator

- bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  inequality operator

13.111.1 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.

13.111.2 Constructor & Destructor Documentation

13.111.2.1 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).

13.111.2.2 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.

13.111.3 Member Function Documentation

13.111.3.1 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::prPairParameters, and ParamResponsePair::prPairResponse.
13.111.3.2  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::prPairParameters, and ParamResponsePair::prPairResponse.

13.111.4  Member Data Documentation

13.111.4.1  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_id(), ParamResponsePair::eval_interface_ids(), ParamResponsePair::interface_id(), ParamResponsePair::operator=(), Dakota::operator==(), ParamResponsePair::read(), ParamResponsePair::read_annotated(), ParamResponsePair::write(), ParamResponsePair::write_annotated(), and ParamResponsePair::write_tabular().

The documentation for this class was generated from the following files:

- ParamResponsePair.hpp
- ParamResponsePair.cpp
### 13.112 ParamStudy Class Reference

Class for vector, list, centered, and multidimensional parameter studies. Inheritance diagram for ParamStudy::

```
ParamStudy
  |       |
  |       |  Analyzer
  |       |  PStudyDACE
  |       |  Iterator
```

**Public Member Functions**

- **ParamStudy** (Model &model)
  
  *constructor*

- **~ParamStudy** ()
  
  *destructor*

- **void pre_run ()**
  
  *pre-run portion of run_iterator (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- **void extract_trends ()**
  
  *Redefines the run_iterator virtual function for the PStudy/DACE branch.*

- **void post_input ()**
  
  *read tabular data for post-run mode*

- **void post_run (std::ostream &s)**
  
  *post-run portion of run_iterator (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 continous/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, bool annotated)

• template<typename OrdinalType, typename ScalarTypeA, typename ScalarTypeC, typename ScalarTypeDI, 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, ScalarTypeDR> &dr_data)
  distributes incoming all vector in standard variable ordering among continuous, discrete int, and discrete real vectors

• template<typename OrdinalType, typename ScalarType>
  bool distribute(const std::vector<ScalarType> &all_data,
                  std::vector<ScalarType> &c_data,
                  std::vector<ScalarType> &di_data)
  distributes incoming all array in standard variable ordering among continuous, discrete int, and discrete real arrays

• bool distribute_list_of_points(const RealVector &list_of_pts)
  distributes incoming list_of_pts among listCVPoints, listDIVPoints, 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,discReal}VarPartitions and global bounds

• bool check_num_steps(int num_steps)
  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 &dr_steps)
  sanity check for centered parameter study

- bool check_sets (const IntVector &c_steps, const IntVector &di_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 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

- short pStudyType
  internal code for parameter study type: LIST, VECTOR_SV, VECTOR_FP, CENTERED, or MULTIDIM

- size_t numEvals
  total number of parameter study evaluations computed from specification

- RealVectorArray listCVPoints
  array of continuous evaluation points for the list_parameter_study

- IntVectorArray listDIVPoints
  array of discrete int evaluation points for the list_parameter_study
• RealVectorArray listDRVPoints
  array of discrete real evaluation points for the list_parameter_study

• RealVector initialCVPoint
  the continuous starting point for vector and centered parameter studies

• IntVector initialDIVPoint
  the continuous starting point for vector and centered parameter studies

• RealVector initialDRVPoint
  the continuous starting point for vector and centered parameter studies

• RealVector contStepVector
  the n-dimensional continuous increment

• IntVector discIntStepVector
  the n-dimensional discrete value or index increment

• IntVector discRealStepVector
  the n-dimensional discrete real index increment

• RealVector finalPoint
  the ending point for vector_parameter_study (a specification option)

• int numSteps
  the number of times continuous/discrete step vectors are applied for vector_parameter_study (a specification option)

• IntVector contStepsPerVariable
  number of offsets in the plus and the minus direction for each continuous variable in a centered_parameter_study

• IntVector discIntStepsPerVariable
  number of offsets in the plus and the minus direction for each discrete integer variable in a centered_parameter_study

• IntVector discRealStepsPerVariable
  number of offsets in the plus and the minus direction for each discrete real variable in a centered_parameter_study

• UShortArray contVarPartitions
  number of partitions for each continuous variable in a multidim_parameter_study

• UShortArray discIntVarPartitions
  number of partitions for each discrete integer variable in a multidim_parameter_study

• UShortArray discRealVarPartitions
  number of partitions for each discrete real variable in a multidim_parameter_study
13.112.1 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.

13.112.2 Member Function Documentation

13.112.2.1 void pre_run () [virtual]

pre-run portion of run_iterator (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 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(), ParamStudy::distribute_partitions(), ParamStudy::final_point_to_step_vector(), ParamStudy::finalPoint, ParamStudy::initialCVPoint, ParamStudy::initialDIVPoint, ParamStudy::initialDRVPoint, Iterator::iteratedModel, ParamStudy::multidim_loop(), ParamStudy::numEvals, ParamStudy::numSteps, Iterator::outputLevel, ParamStudy::pStudyType, ParamStudy::sample(), Variables::shared_data(), ParamStudy::vector_loop(), Dakota::write_data(), and Dakota::write_ordered().

13.112.2.2 void post_run (std::ostream &s) [virtual]

post-run portion of run_iterator (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 Analyzer::allResponses, Analyzer::allVariables, SensAnalysisGlobal::compute_correlations(), PStudyDACE::pStudyDACESensGlobal, ParamStudy::pStudyType, and Iterator::subIteratorFlag.
13.112.2.3  bool load_distribute_points (const String & points_filename, bool annotated)  [private]

Load from file and distribute points; using this function to manage construction of the temporary array

References ParamStudy::distribute_list_of_points(), Iterator::numContinuousVars, Iterator::numDiscreteIntVars,
Iterator::numDiscreteRealVars, and Dakota::read_data_tabular().

Referenced by ParamStudy::ParamStudy().

The documentation for this class was generated from the following files:

- ParamStudy.hpp
- ParamStudy.cpp
13.113 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

13.113.1 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
13.114 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

13.114.1 Detailed Description

wrapper to delegate to the ParamResponsePair hash_value function

The documentation for this struct was generated from the following file:

- PRPMultiIndex.hpp
13.115 PecosApproximation Class Reference

Derived approximation class for global basis polynomials. Inheritance diagram for PecosApproximation:

```
Approximation
   ↓
PecosApproximation
```

Public Member Functions

- **PecosApproximation ()**  
  default constructor

- **PecosApproximation (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  alternate constructor

- **PecosApproximation (ProblemDescDB &problem_db, size_t num_vars)**  
  standard ProblemDescDB-driven constructor

- **~PecosApproximation ()**  
  destructor

- **void increment_order ()**
  increment OrthogPolyApproximation::approxOrder uniformly

- **void update_order (const UShortArray &order)**
  replace OrthogPolyApproximation::approxOrder with incoming order

- **void solution_approach (short soln_approach)**
  set pecosBasisApprox.configOptions.expCoeffsSolnApproach

- **short solution_approach () const**
  get pecosBasisApprox.configOptions.expCoeffsSolnApproach

- **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
• bool expansion_gradient_flag () const
  get pecosBasisApprox.configOptions.expansionGradFlag

• void vbd_flag (bool flag)
  set pecosBasisApprox.configOptions.vbdFlag

• bool vbd_flag () const
  get pecosBasisApprox.configOptions.vbdFlag

• void vbd_order_limit (unsigned short vbd_order)
  set pecosBasisApprox.configOptions.vbdOrderLimit

• unsigned short vbd_order_limit () const
  get pecosBasisApprox.configOptions.vbdOrderLimit

• void refinement_control (short refine_cntl)
  set pecosBasisApprox.configOptions.refinementControl

• short refinement_control () const
  get pecosBasisApprox.configOptions.refinementControl

• void maximum_iterations (int max_iter)
  set pecosBasisApprox.configOptions.maxIterations

• int maximum_iterations () const
  get pecosBasisApprox.configOptions.maxIterations

• void convergence_tolerance (Real conv_tol)
  set pecosBasisApprox.configOptions.convergenceTol

• Real convergence_tolerance () const
  get pecosBasisApprox.configOptions.convergenceTol

• 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::BitArrayULongMap & sobol_index_map () const
  return polyApproxRep->sobolIndexMap

• const Pecos::RealVector & sobol_indices () const
  return polyApproxRep->sobolIndices
• const Pecos::RealVector & total_sobol_indices () const
  return polyApproxRep->totalSobolIndices

• const Pecos::RealVector & dimension_decay_rates () const
  return OrthogPolyApproximation::decayRates

• 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, const
  Pecos::BasisConfigOptions &bc_options)
  invoke Pecos::OrthogPolyApproximation::construct_basis()

• void basis_types (const Pecos::ShortArray &basis_types)
  set Pecos::OrthogPolyApproximation::basisTypes

• const Pecos::ShortArray & basis_types () const
  get Pecos::OrthogPolyApproximation::basisTypes

• void polynomial_basis (const std::vector<Pecos::BasisPolynomial> &poly_basis)
  set Pecos::OrthogPolyApproximation::polynomialBasis

• const std::vector<Pecos::BasisPolynomial> & polynomial_basis () const
  get Pecos::OrthogPolyApproximation::polynomialBasis

• void cross_validation (bool flag)
  invoke Pecos::OrthogPolyApproximation::cross_validation()

• void coefficients_norms_flag (bool flag)
  invoke Pecos::OrthogPolyApproximation::coefficients_norms_flag()

• size_t expansion_terms () const
  return Pecos::OrthogPolyApproximation::expansion_terms()

• const UShortArray & expansion_order () const
  return Pecos::OrthogPolyApproximation::expansion_order()

• void noise_tolerance (const RealVector &noise_tol)
  set the noise tolerance(s) for compressed sensing algorithms

• void l2_penalty (Real l2_pen)
  set the L2 penalty parameter for LASSO (elastic net variant)
• 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 &x, 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 &x, 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 &x, 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 &x, bool cdf_flag, Real beta_bar)
  
  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** ()
  
  compute moments up to the order supported by the Pecos polynomial approximation

- void **compute_moments** (const Pecos::RealVector &x)
  
  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_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

- Pecos::BasisApproximation & **pecos_basis_approximation** ()
  
  return pecosBasisApprox
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 ()
  builds the approximation from scratch

- void rebuild ()
  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 restore ()
  restores state prior to previous append()

- bool restore_available ()
  queries availability of restoration for trial set

- size_t restoration_index ()
  return index of trial set within restorable bookkeeping sets

- void finalize ()
  finalize approximation by applying all remaining trial sets

- size_t finalization_index (size_t i)
  return index of i-th trailing trial set within restorable bookkeeping sets

- void store ()
  store current approximation for later combination

- void combine (short corr_type)
  combine current approximation with previously stored approximation

- void print_coefficients (std::ostream &s, bool normalized=false)
**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*

### 13.115.1 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.

### 13.115.2 Member Function Documentation

#### 13.115.2.1 void **build** () [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 PecosApproximation::pecosBasisApprox.
13.115.2.2  void rebuild()  [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.

13.115.2.3  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.

13.115.2.4  void restore()  [inline, protected, virtual]

restores state prior to previous append(). 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.

13.115.2.5  void finalize()  [inline, protected, virtual]

finalizes 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 PecosApproximation::pecosBasisApprox.

The documentation for this class was generated from the following files:

- PecosApproximation.hpp
- PecosApproximation.cpp
13.116 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 manage_inputs (CommandLineHandler &cmd_line_handler)**
  - invokes `manage_inputs(const char*, ...)` using the dakota input filename passed with the "-input" option on the DAKOTA command line. This is the normal API employed in main.cpp.

- **void manage_inputs (const char *dakota_input_file, const char *parser_options=NULL, bool echo_input=true, void(*callback)(void*)=NULL, void*callback_data=NULL)**
  - invokes parse_inputs() to populate the problem description database and execute any callback function, broadcast() to propagate DB data to all processors, and post_process() to construct default variables/response vectors. This is an alternate API used by the file parsing mode in library_mode.cpp.

- **void parse_inputs (const char *dakota_input_file, const char *parser_options=NULL, bool echo_input=true, void(*callback)(void*)=NULL, void*callback_data=NULL)**
  - parses the input file and populates the problem description database. This function reads from the dakota input filename passed in and allows subsequent modifications to be done by a callback function. This API is used by the mixed mode option in library_mode.cpp since it allows broadcast() and post_process() to be deferred until all inputs have been provided.

- **void check_input (const char *dakota_input_file=NULL, bool echo_input=true)**
verifies that there is at least one of each of the required keywords in the dakota input file. Used by `parse_inputs()`.

- **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 (const 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 ()**
  
  For a (default) strategy lacking a 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 (const 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 (const 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 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 SizetArray & get_sza (const String &entry_name) const
  get an SizetArray out of the database based on an identifier string

• const UShortArray & get_usa (const String &entry_name) const
  get a 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 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 RealRealMapArray & get_rrma (const String &entry_name) const
  get a RealRealMapArray 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 DataStrategy &data_strategy)` 
  set the `DataStrategy` 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 (const 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 RealSymMatrix &rsm)` 
  set a `RealMatrix` 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 RealRealMapArray &rrma)`
set a RealRealMapArray within the database 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

- bool is_null () const
  function to check dbRep (does this envelope contain a letter)

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 char ∗dakota_input_file, const char ∗parser_options)
  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

- DataStrategy strategySpec
  the strategy specification (only one allowed) resulting from a call to strategy_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()

- 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 strategyCntr
  counter for strategy specifications used in check_input

Private Member Functions

• const Iterator & get_iterator (Model &model)
  retrieve an existing Iterator, if it exists, 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 (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 strategySpec 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 strategySpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage_inputs().

• void echo_input_file (const char *dakota_input_file)
  echo the (potentially) specified input file 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

- 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

Friends

- class Model
  Model requires access to get_variables() and get_response().

- class SingleModel
  SingleModel 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 Strategy
  Strategy requires access to get_iterator().

- class SingleMethodStrategy
  SingleMethodStrategy requires access to get_model().

- class HybridStrategy
  HybridStrategy requires access to get_model().

- class SequentialHybridStrategy
  SequentialStrategy requires access to get_iterator().

- class ConcurrentStrategy
  ConcurrentStrategy requires access to get_model().

- class SurrBasedLocalMinimizer
  SurrBasedLocalMinimizer requires access to get_iterator().

- class SurrBasedGlobalMinimizer
  SurrBasedGlobalMinimizer requires access to get_iterator().
13.116 ProblemDescDB Class Reference

13.116.1 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 (DataStrategy, DataMethod, DataVariables, DataInterface, or DataResponses) and, for all cases except strategy, appends the object to a linked list (dataMethodList, dataVariablesList, dataInterfaceList, or dataResponsesList). No strategy linked list is used since only one strategy specification is allowed.

13.116.2 Constructor & Destructor Documentation

13.116.2.1 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.

13.116.2.2 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 BaseConstructor builds the actual base class data inherited by the derived classes. References Dakota::abort_handler(), ProblemDescDB::dbRep, and ProblemDescDB::get_db().

13.116.2.3 ProblemDescDB (const ProblemDescDB & db)


13.116.2.4 ~ProblemDescDB ()

destructor Destructor decrements referenceCount and only deletes dbRep when referenceCount reaches zero. References Dakota::Dak_pddb, ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

13.116.2.5 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).
13.116.3 Member Function Documentation

13.116.3.1 ProblemDescDB operator= (const ProblemDescDB & db)

Assignment operator. Assignment operator decrements referenceCount for old dbRep, assigns new dbRep, and
increments referenceCount for new dbRep.
References ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

13.116.3.2 void manage_inputs (CommandLineHandler & cmd_line_handler)

Invokes manage_inputs(const char*, ...) using the dakota input filename passed with the "-input" option on the
DAKOTA command line. This is the normal API employed in main.cpp. Manage command line inputs using the
CommandLineHandler class and parse the input file.
References ProblemDescDB::dbRep, ProblemDescDB::manage_inputs(), ProblemDescDB::parallelLib, GetLongOpt::retrieve(), and ParallelLibrary::world_rank().
Referenced by main(), ProblemDescDB::manage_inputs(), run_dakota(), and run_dakota_parse().

13.116.3.3 void manage_inputs (const char * dakota_input_file, const char * parser_options = NULL, bool echo_input = true, void (*)(void *) callback = NULL, void * callback_data = NULL)

Invokes parse_inputs() to populate the problem description database and execute any callback function, broadcast() to propagate DB data to all processors, and post_process() to construct default variables/response vectors. This is an alternate API used by the file parsing mode in library_mode.cpp. Parse the input file, broadcast it to all processors, and post-process the data on all processors.
References ProblemDescDB::broadcast(), ProblemDescDB::dbRep, ProblemDescDB::manage_inputs(), ProblemDescDB::parse_inputs(), and ProblemDescDB::post_process().

13.116.3.4 void parse_inputs (const char * dakota_input_file, const char * parser_options = NULL, bool echo_input = true, void (*)(void *) callback = NULL, void * callback_data = NULL)

 Parses the input file and populates the problem description database. This function reads from the dakota input filename passed in and allows subsequent modifications to be done by a callback function. This API is used by the mixed mode option in library_mode.cpp since it allows broadcast() and post_process() to be deferred until all inputs have been provided. Parse the input file, execute the callback function (if present), and perform basic checks on keyword counts.
References ProblemDescDB::check_input(), ProblemDescDB::dbRep, ProblemDescDB::derived_parse_inputs(), ProblemDescDB::parallelLib, ProblemDescDB::parse_inputs(), and ParallelLibrary::world_rank().
Referenced by ProblemDescDB::manage_inputs(), ProblemDescDB::parse_inputs(), and run_dakota_mixed().

13.116.3.5 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::manage_inputs(), Dakota::run_dakota_data(), and run_dakota_mixed().

13.116.3.6 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
13.117 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:

```
  Interface
  |________|
  |        |
  v        v
ApplicationInterface
  |________|
  |        |
  v        v
ProcessApplicInterface
  |________|
  |        |
  v        v
ProcessHandleApplicInterface
  |________|
  |        |
  v        v
SysCallApplicInterface
  |________|
  |        |
  v        v
ForkApplicInterface
  |________|
  |        |
  v        v
SpawnApplicInterface
  |________|
  |        |
  v        v
GridApplicInterface
```

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`
  clean up any interface parameter/response files when aborting

- `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

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

• 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)

- 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)

- 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::string curWorkdir
  working directory when useWorkdir is true

- std::map<int, std::pair<std::string, std::string>> fileNameMap
  stores parameters and results file names used in spawning function evaluations. Map key is the function evaluation identifier.

- bool useWorkdir
  whether to use a new or specified work_directory

- std::string workDir
  its name, if specified...

- bool dirTag
  whether to tag the working directory

- bool dirSave
  whether dir_save was specified

- bool dirDel
  whether to delete the directory when Dakota terminates

- bool haveWorkdir
  for dirTag, whether we have workDir

- std::string templateDir
template directory (if specified)

- StringArray templateFiles
  template files (if specified)

- bool templateCopy
  whether to force a copy (versus link) every time

- bool templateReplace
  whether to replace existing files

- bool haveTemplateDir
  state variable for template directory

- std::string dakDir
  Dakota directory (if needed).

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

Private Attributes

- String2DArray analysisComponents
  the set of optional analysis components used by the analysis drivers (from the analysis_components interface specification)

13.117.1 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.

13.117.2 Member Function Documentation

13.117.2.1 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().

Dakota Version 5.4 Developers Manual generated on November 7, 2013
Referenced by ProcessHandleApplicInterface::create_evaluation_process().

The documentation for this class was generated from the following files:

- ProcessApplicInterface.hpp
- ProcessApplicInterface.cpp
13.118 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_iterator_concurrency)
- void **set_communicators_checks** (int max_iterator_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 asynch evaluations

• virtual void join_analysis_process_group (bool new_group)
  create (if new_group) and join the process group for asynch 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_evaluations.

• 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

Protected Attributes

• std::map< pid_t, int > evalProcessIdMap
  map of fork process id’s to function evaluation id’s for asynchronous evaluations
13.118 ProcessHandleApplicInterface Class Reference

- 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().

13.118.1 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).

13.118.2 Member Function Documentation

13.118.2.1 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 convenience function for ProcessHandleApplicInterface::synchronous_local_analyses() below.

Reimplemented from ApplicationInterface.

References ProcessHandleApplicInterface::create_analysis_process(), and ProcessHandleApplicInterface::driver_argument_list().

13.118.2.2 void init_communicators_checks (int max_iterator_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().

13.118.2.3 void set_communicators_checks (int max_iterator_concurrency) [inline, protected, virtual]

Process run-time issues as hard errors.

Reimplemented from ApplicationInterface.

References Dakota::abort_handler(), ApplicationInterface::check_multiprocessor_analysis(), and ApplicationInterface::check_multiprocessor_asynchronous().
13.118.2.4 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_async() 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::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, ProcessHandleApplicInterface::parallelLib, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, ProcessHandleApplicInterface::serve_analyses_async(), ApplicationInterface::serve_analyses_synch(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::synchronous_local_analyses().

13.118.2.5 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 (WIFEXITED(status)==0) or if either execvp or the application returned a status code of -1 (WIFEXITED(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().

References Dakota::abort_handler(), ForkApplicInterface::create_analysis_process(), SpawnApplicInterface::test_local_analyses_send(), SpawnApplicInterface::test_local_evaluations(), ForkApplicInterface::wait(), SpawnApplicInterface::wait_local_analyses(), and SpawnApplicInterface::wait_local_evaluations().

13.118.2.6 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.

References Dakota::abort_handler(), ProcessHandleApplicInterface::analysisProcessIdMap, Application
Interface::asynchLocalAnalysisConcurrency, ProcessHandleApplicInterface::create_analysis_process(),
ProcessHandleApplicInterface::driver_argument_list(), ApplicationInterface::numAnalysisDrivers, and
ProcessHandleApplicInterface::wait_local_analyses().

Referenced by ProcessHandleApplicInterface::create_evaluation_process().

13.118.2.7 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, Application-
Interface::asynchLocalAnalysisConcurrency, ProcessHandleApplicInterface::create_analysis_process(),
ProcessHandleApplicInterface::driver_argument_list(), ParallelLibrary::irecv_ea(), ApplicationInter-
face::numAnalysisDrivers, ApplicationInterface::parallelLib, ParallelLibrary::recv_ea(), ParallelLibrary::test(),
and ProcessHandleApplicInterface::test_local_analyses_send().

Referenced by ProcessHandleApplicInterface::create_evaluation_process().

The documentation for this class was generated from the following files:

- ProcessHandleApplicInterface.hpp
- ProcessHandleApplicInterface.cpp
13.119  PStudyDACE Class Reference

Base class for managing common aspects of parameter studies and design of experiments methods. Inheritance diagram for PStudyDACE::

```
     Iterator
      |
    Analyzer
     |
PStudyDACE
```

```
DDACEDesignCompExp  FSUDesignCompExp  ParamStudy  PSUADEDesignCompExp
```

**Protected Member Functions**

- **PStudyDACE (Model &model)**
  
  constructor

- **PStudyDACE (NoDBBaseConstructor, Model &model)**
  
  alternate constructor for instantiations "on the fly"

- **~PStudyDACE ()**
  
  destructor

- **void run ()**
  
  run portion of run_iterator; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- **void print_results (std::ostream &s)**
  
  print the final iterator results

- **virtual void extract_trends ()=0**
  
  Redefines the run_iterator virtual function for the PStudy/DACE branch.

- **void volumetric_quality (int ndim, int num_samples, double *sample_points)**
  
  Calculation of volumetric quality measures.

**Protected Attributes**

- **SensAnalysisGlobal pStudyDACESensGlobal**
  
  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

13.119.1 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.

13.119.2 Member Function Documentation

13.119.2.1 void run () [inline, protected, virtual]

run portion of run_iterator; 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::bestVarsRespMap, and PStudyDACE::extract_trends().

13.119.2.2 void print_results (std::ostream & s) [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(), PStudyDACE::dMeas, PStudyDACE::hMeas, Iterator::iteratedModel, Analyzer::numLSqTerms, Analyzer::numObjFns, SensAnalysisGlobal::print_correlations(), Analyzer::print_sobol_indices(), PStudyDACE::pStudyDACESensGlobal, Model::response_labels(), PStudyDACE::tauMeas, PStudyDACE::varBasedDecompFlag, and PStudyDACE::volQualityFlag.

13.119.2.3  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
13.120  PSUADEDesignCompExp Class Reference

Wrapper class for the PSUADE library. Inheritance diagram for PSUADEDesignCompExp:

```
PSUADEDesignCompExp
  PStudyDACE
    Analyzer
      Iterator
```

Public Member Functions

- **PSUADEDesignCompExp (Model &model)**
  *primary constructor for building a standard DACE iterator*

- **~PSUADEDesignCompExp ()**
  *destructor*

- **void pre_run ()**
  *pre-run portion of run_iterator (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 extract_trends ()**
  *Redefines the run_iterator virtual function for the PStudy/DACE branch.*

- **void post_run (std::ostream &s)**
  *post-run portion of run_iterator (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**
  *get the current number of samples*

- **void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)**
  *reset sampling iterator to use at least min_samples*

- **const String & 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)
Returns one block of samples (ndim * num_samples).

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 run() 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
13.120.1 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.

13.120.2 Constructor & Destructor Documentation

13.120.2.1 PSUADEDesignCompExp (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::maxConcurrency, Iterator::methodName, and PSUADEDesignCompExp::numSamples.

13.120.3 Member Function Documentation

13.120.3.1 void pre_run () [virtual]

pre-run portion of run_iterator (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 PSUADEDesignCompExp::get_parameter_sets(), and Iterator::iteratedModel.

13.120.3.2 void post_run (std::ostream & s) [virtual]

post-run portion of run_iterator (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 Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Iterator::iteratedModel, Iterator::numContinuousVars, Iterator::numFunctions, and PSUADEDesignCompExp::numSamples.
13.120.3.3 int num_samples () const [inline, virtual]

get the current number of samples 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 maxConcurrency. May be (is) overridden by derived classes.

Reimplemented from Iterator.

References PSUADEDesignCompExp::numSamples.

13.120.3.4 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, Iterator::numContinuousVars, PSUADEDesignCompExp::numPartitions, PSUADEDesignCompExp::numSamples, and PSUADEDesignCompExp::varPartitionsSpec.

Referenced by PSUADEDesignCompExp::get_parameter_sets().

The documentation for this class was generated from the following files:

- PSUADEDesignCompExp.hpp
- PSUADEDesignCompExp.cpp
13.121 PythonInterface Class Reference

Inheritance diagram for PythonInterface:

```
    Interface
      |
      v
ApplicationInterface
      |
      v
DirectApplicInterface
      |
      v
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

13.121.1 Detailed Description

Specialization of DirectApplicInterface to link to Python analysis drivers. Includes convenience functions to map data to/from Python

13.121.2 Member Function Documentation

13.121.2.1 int derived_map_ac (const String &ae_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().
13.121.2.2 bool python_convert_int (const ArrayT & src, Size sz, PyObject ** dst) [inline, 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
13.122 RecastBaseConstructor Struct Reference

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

Public Member Functions

- `RecastBaseConstructor(int=0)`
  
  C++ structs can have constructors.

13.122.1 Detailed Description

Dummy struct for overloading constructors used in on-the-fly Model instantiations. RecastBaseConstructor 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
13.123 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::

```
Model
  RecastModel
```

Public Member Functions

  *standard constructor*

- **RecastModel** (const Model &sub_model, const SizetArray &vars_comps_total, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset)
  *alternate constructor*

- **~RecastModel** ()
  *destructor*

  *completes initialization of the RecastModel after alternate construction*

- void **inverse_mappings** (void(*inv_vars_map)(const Variables &sub_model_vars, Variables &recast_vars), void(*inv_set_map)(const Variables &sub_model_vars, const ActiveSet &recast_set, const ActiveSet &sub_model_set), void(*inv_pri_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_response), void(*inv_sec_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_response))

Dakota Version 5.4 Developers Manual generated on November 7, 2013
provide optional inverse mappings

- **void transform_variables** (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 inverse_transform_variables** (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 &sub_model_resp, Response &recast_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**

- **void derived_compute_response** (const ActiveSet &set)
  portion of compute_response() specific to RecastModel (forward to subModel.compute_response())

- **void derived_asynch_compute_response** (const ActiveSet &set)
  portion of asynch_compute_response() specific to RecastModel (forward to subModel.asynch_compute_response())

- **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 (bool recurse_flag=true)**
  pass request to subModel if recursing and then update from it

- **Interface & interface ()**
  return subModel interface

- **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)

- **void build_approximation ()**
  builds the subModel approximation

- **bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)**
  builds the subModel approximation

- **void update_approximation (bool rebuild_flag)**
  replaces data in the subModel approximation

- **void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)**
  replaces data in the subModel approximation

- **void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)**
  replaces data in the subModel approximation

- **void append_approximation (bool rebuild_flag)**
  appends data to the subModel approximation
- void **append_approximation** (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  
  appends data to the subModel approximation

- void **append_approximation** (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  
  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 **restore_approximation**()

- void **restore_approximation** ()
  
  restore a previous approximation data state within a surrogate

- bool **restore_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** ()
  
  move the current approximation into storage for later combination

- void **combine_approximation** (short corr_type)
  
  combine the current approximation with one previously stored

- std::vector< **Approximation** > & **approximations** ()
  
  retrieve the set of Approximations from the subModel

- const RealVectorArray & **approximation_coefficients** ()
  
  retrieve the approximation coefficients from the subModel

- void **approximation_coefficients** (const RealVectorArray &approx_coeffs)
  
  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.
• String 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)

• void derived\_init\_communicators (int max\_iterator\_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 (int max\_iterator\_concurrency, bool recurse\_flag=true)
  
  set active parallel configuration within subModel

• void derived\_free\_communicators (int max\_iterator\_concurrency, bool recurse\_flag=true)
  
  deallocate communicator partitions for the RecastModel (request forwarded to subModel)

• void serve (int max\_iterator\_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

• int evaluation\_id () const
  
  return the current evaluation id for the RecastModel (request forwarded to subModel)

• 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)
• virtual void eval_tag_prefix (const String &eval_id_str)
  set the hierarchical eval ID tag prefix

Private Member Functions

• void initialize_data_from_submodel ()
  code shared among constructors to initialize base class data from submodel

• void update_from_sub_model ()
  update current variables/labels/bounds/targets from subModel

Private Attributes

• Model subModel
  the sub-model underlying the function pointers

• 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.

• IntActiveSetMap recastSetMap
map of recast active set passed to derived_asynch_compute_response(). Needed for currentResponse update in synchronization routines.

- IntVariablesMap recastVarsMap
  map of recast variables used by derived_asynch_compute_response(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.

- IntVariablesMap subModelVarsMap
  map of subModel variables used by derived_asynch_compute_response(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.

- IntResponseMap recastResponseMap
  map of recast responses used by RecastModel::derived_synchronize() and RecastModel::derived_synchronize_nowait()

- 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()
13.123.1 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. This is currently used to recast SBO approximate subproblems, but can be used for multiobjective, input/output scaling, and other problem modifications in the future.

13.123.2 Constructor & Destructor Documentation

13.123.2.1 RecastModel (const Model & sub_model, const Sizet2DArray & vars_map_indices, const SizetArray & vars_comps_totals, 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, size_t recast_secondary_offset, 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 constructor Default recast model constructor. Requires full definition of the transformation. 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(), Constraints::copy(), Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Variables::cv(), Response::function_gradients(), Response::function_hessians(), RecastModel::initialize_data_from_submodel(), RecastModel::nonlinearRespMapping, Constraints::num_functions(), Constraints::num_linear_eq_constraints(), Constraints::num_linear_ineq_constraints(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Model::numDerivVars, Model::numFns, RecastModel::primaryRespMapIndices, RecastModel::primaryRespMapping, Constraints::reshape(), Response::reshape(), RecastModel::respMapping, RecastModel::secondaryRespMapIndices, RecastModel::secondaryRespMapping, RecastModel::subModel, Model::user_defined_constraints(), Variables::variables_components_totals(), RecastModel::variablesMapping, and Variables::view().

13.123.2.2 RecastModel (const Model & sub_model, const SizetArray & vars_comps_totals, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset)

alternate constructor 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 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 Constraints::copy(), Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Variables::cv(), Response::function_gradients(), Response::function_hessians(), RecastModel::initialize_data_from_submodel().
submodel(), Model::num_functions(), Constraints::num_linear_eq_constraints(), Constraints::num_linear_ineq_constraints(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Model::numDerivVars, Model::numFns, Constraints::reshape(), Response::reshape(), RecastModel::subModel, Model::user_defined_constraints(), Model::userDefinedConstraints, Variables::variables_components_totals(), and Variables::view.

13.123.3 Member Function Documentation

13.123.3.1 void initialize (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

completes initialization of the RecastModel 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 EffGlobalMinimizer::minimize_surrogates_on_model(), NonDLocalReliability::mpp_search(), NonDGlobalReliability::optimize_gaussian_process(), NonDLocalInterval::quantify_uncertainty(), NonDGlobalInterval::quantify_uncertainty(), and Minimizer::scale_model().

13.123.3.2 void derived_compute_response (const ActiveSet & set) [protected, virtual]

portion of compute_response() specific to RecastModel (forward to subModel.compute_response()) 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::compute_response(), Model::current_response(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Model::parallel_configuration_iterator(), ParallelLibrary::parallel_configuration_iterator(), Model::parallelLib, RecastModel::respMapping, RecastModel::subModel, RecastModel::transform_response(), RecastModel::transform_set(), RecastModel::transform_variables(), and Response::update().

13.123.3.3 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.

13.123.3.4 void update_from_sub_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 Model::aleatDistParams, Model::aleatory_distribution_parameters(), Model::continuous_lower_bounds(), Constraints::continuous_lower_bounds(), Model::continuous_upper_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_variable_labels(), Variables::continuous_variables(), Variables::continuous_variable_labels(), Model::currentResponse, Model::currentVariables, Model::discrete_design_set_int_values(), Model::discrete_design_set_real_values(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_lower_bounds(), Model::discrete_int_upper_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_variable_labels(), Variables::discrete_int_variable_labels(), Model::discrete_int_variables(), Variables::discrete_int_variables(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_upper_bounds(), Constraints::discrete_real_upper_bounds(), Variables::discrete_real_variable_labels(), Variables::discrete_real_variable_labels(), Model::discrete_real_variables(), Variables::discrete_real_variables(), Model::discrete_state_set_int_values(), Model::discrete_state_set_real_values(), Model::discrete_variable_labels(), Variables::discrete_variable_labels(), Model::discrete_variables(), Variables::discrete_variables(), Model::epistDistParams, Model::epistemic_distribution_parameters(), Variables::function_label(), Model::inactive_continuous_lower_bounds(), Constraints::inactive_continuous_lower_bounds(), Model::inactive_continuous_upper_bounds(), Constraints::inactive_continuous_upper_bounds(), Variables::inactive_continuous_variable_labels(), Model::inactive_continuous_variables(), Variables::inactive_continuous_variables(), Model::inactive_continuous_variable_labels(), Variables::inactive_continuous_variable_labels(), Constraints::inactive_continuous_int_lower_bounds(), Model::inactive_continuous_int_lower_bounds(), Variables::inactive_continuous_int_variable_labels(), Variables::inactive_continuous_int_variable_labels(), Constraints::inactive_continuous_int_upper_bounds(), Model::inactive_continuous_int_upper_bounds(), Variables::inactive_continuous_int_variables(), Variables::inactive_continuous_int_variables(), Constraints::inactive_discrete_int_lower_bounds(), Model::inactive_discrete_int_lower_bounds(), Variables::inactive_discrete_int_variable_labels(), Variables::inactive_discrete_int_variable_labels(), Constraints::inactive_discrete_int_upper_bounds(), Model::inactive_discrete_int_upper_bounds(), Variables::inactive_discrete_int_variables(), Variables::inactive_discrete_int_variables(), Constraints::inactive_discrete_real_lower_bounds(), Model::inactive_discrete_real_lower_bounds(), Variables::inactive_discrete_real_variable_labels(), Variables::inactive_discrete_real_variable_labels(), Constraints::inactive_discrete_real_upper_bounds(), Model::inactive_discrete_real_upper_bounds(), Variables::inactive_discrete_real_variables(), Variables::inactive_discrete_real_variables(), Model::linear_eq_constraint_coeffs(), Constraints::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Constraints::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), Constraints::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), Constraints::linear_ineq_constraint_upper_bounds(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::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::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primaryRespFnSense, Model::primaryRespFnWts, RecastModel::primaryRespMapping, Model::response_labels(), RecastModel::secondaryRespMapping, RecastModel::subModel, Model::userDefinedConstraints, and RecastModel::variablesMapping.

Referenced by RecastModel::update_from_subordinate_model().

The documentation for this class was generated from the following files:
• RecastModel.hpp
• RecastModel.cpp
13.124  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

Protected Member Functions

- **void reshape** (const SizetArray &vc_totals)
  
  reshape the lower/upper bound arrays within the Constraints hierarchy

- **void build_active_views** ()
  
  construct active views of all variables bounds arrays

- **void build_inactive_views** ()
  
  construct inactive views of all variables bounds arrays
13.124 RelaxedVarConstraints Class Reference

13.124.1 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).

13.124.2 Constructor & Destructor Documentation

13.124.2.1 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.

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::build_views(), SharedVariablesData::components_totals(), Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Constraints::manage_linear_constraints(), Dakota::merge_data_partial(), Constraints::sharedVarsData, and SharedVariablesData::vc_lookup().

13.124.3 Member Function Documentation

13.124.3.1 void reshape (const SizetArray & vc_totals) [protected, virtual]

reshape the lower/upper bound arrays within the Constraints hierarchy Resizes the derived bounds arrays.

Reimplemented from Constraints.

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, and Constraints::build_views().

Referenced by RelaxedVarConstraints::RelaxedVarConstraints().

The documentation for this class was generated from the following files:

- RelaxedVarConstraints.hpp
- RelaxedVarConstraints.cpp
13.125 RelaxedVariables Class Reference

Derived class within the Variables hierarchy which employs the relaxation of discrete variables. Inheritance diagram for RelaxedVariables::

```
    Variables
     |
     v
    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) const
  - write a variables object to an std::ostream

- `void write_aprepro` (std::ostream &s) const
  - write a variables object to an std::ostream in aprepro format

- `void reshape` (const SizetArray &vc_totals)
  - reshapes an existing Variables object based on the incoming variablesComponents

- `void build_active_views`()
  - construct active views of all variables arrays

- `void build_inactive_views`()
13.125 RelaxedVariables Class Reference

13.125.1 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)).

13.125.2 Constructor & Destructor Documentation

13.125.2.1 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: BranchAndBoundOptimizer. Extract fundamental variable types and labels and merge continuous and discrete domains to create aggregate arrays and views.

References Variables::allContinuousVars, Variables::build_views(), SharedVariablesData::components_totals(), Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Dakota::merge_data_partial(), Variables::sharedVarsData, SharedVariablesData::vc_lookup(), and SharedVariablesData::view().

13.125.3 Member Function Documentation

13.125.3.1 void read_tabular (std::istream & s) [protected, virtual]

Presumes variables object is appropriately sized to receive data
Reimplemented from Variables.

References Variables::allContinuousVars, SharedVariablesData::components_totals(), Dakota::read_data_partial_tabular(), and Variables::sharedVarsData.

The documentation for this class was generated from the following files:

- RelaxedVariables.hpp
- RelaxedVariables.cpp
13.126  Response Class Reference

Container class for response functions and their derivatives. Response provides the handle class.

Public Member Functions

- **Response ()**
  
  *default constructor*

- **Response (const Variables &vars, const ProblemDescDB &problem_db)**
  
  *standard constructor built from problem description database*

- **Response (const ActiveSet &set)**
  
  *alternate constructor using limited data*

- **Response (const Response &response)**
  
  *copy constructor*

- **~Response ()**
  
  *destructor*

- **Response operator= (const Response &response)**
  
  *assignment operator*

- **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*

- **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*
• const String & responses_id () const  
  return the response identifier

• 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

• 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 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

• 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 (const int &i) const  
  return the i-th function gradient as a const Real*

• RealVector function_gradient_view (const int &i) const  
  return the i-th function gradient as a SerialDenseVector Teuchos::View (shallow copy) for updating in place

• RealVector function_gradient_copy (const 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

- void function_gradient (const RealVector &function_grad, const int &i)
  set a function gradient

- void function_gradients (const RealMatrix &function_grads)
  set all function gradients

- const RealSymMatrix & function_hessian (size_t i) const
  return the i-th function Hessian

- RealSymMatrix function_hessian_view (size_t i) const
  return the i-th function Hessian as a Teuchos::View (shallow copy) for updating in place

- const RealSymMatrixArray & function_hessians () const
  return all function Hessians

- RealSymMatrixArray function_hessians_view ()
  return all function Hessians as Teuchos::Views (shallow copies) for updating in place

- void function_hessian (const RealSymMatrix &function_hessian, size_t i)
  set a function Hessian

- void function_hessians (const RealSymMatrixArray &function_hessians)
  set all function Hessians

- void read (std::istream &s)
  read a response object 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 read (BiStream &s)
  read a response object from the binary restart stream
• **void write** (BoStream &s) const
  
  write a response object to the binary restart 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** () const
  
  a deep copy for use in history mechanisms

• **int data_size** ()
  
  handle class forward to corresponding body class member function

• **void read_data** (double *response_data)
  
  handle class forward to corresponding body class member function

• **void write_data** (double *response_data)
  
  handle class forward to corresponding body class member function

• **void overlay** (const Response &response)
  
  handle class forward to corresponding body class member function

• **void update** (const Response &response)
  
  Used in place of operator= when only results data updates are desired (functionValues/functionGradients/functionHessians 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 RealSymMatrixArray &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)
rehapes response data arrays

- void reset ()
  handle class forward to corresponding body class member function

- void reset_inactive ()
  handle class forward to corresponding body class member function

- bool is_null () const
  function to check responseRep (does this handle contain a body)

Private Attributes

- ResponseRep * responseRep
  pointer to the body (handle-body idiom)

Friends

- bool operator==(const Response &resp1, const Response &resp2)
  equality operator

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

13.126.1 Detailed Description

Container class for response functions and their derivatives. Response provides the handle class. The Response class is a container class for an abstract set of functions (functionValues) and their first (functionGradients) 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). It is not currently part of a class hierarchy, since the abstraction has been sufficiently general and has not required specialization. For memory efficiency, it employs the "handle-body idiom" approach to reference counting and representation sharing (see Coplien "Advanced C++", p. 58), for which Response serves as the handle and ResponseRep serves as the body.

13.126.2 Constructor & Destructor Documentation

13.126.2.1 Response ()

default constructor Need a populated problem description database to build a meaningful Response object, so set the responseRep=NULL in default constructor for efficiency. This then requires a check on NULL in the copy constructor, assignment operator, and destructor.
The documentation for this class was generated from the following files:

- DakotaResponse.hpp
- DakotaResponse.cpp
### 13.127 ResponseRep Class Reference

Container class for response functions and their derivatives. `ResponseRep` provides the body class.

#### Private Member Functions

- **ResponseRep ()**
  - default constructor

- **ResponseRep (const Variables &vars, const ProblemDescDB &problem_db)**
  - standard constructor built from problem description database

- **ResponseRep (const ActiveSet &set)**
  - alternate constructor using limited data

- **~ResponseRep ()**
  - destructor

- **void read (std::istream &s)**
  - read a responseRep object from an std::istream

- **void write (std::ostream &s) const**
  - write a responseRep object to an std::ostream

- **void read_annotated (std::istream &s)**
  - read a responseRep object from an std::istream (annotated format)

- **void write_annotated (std::ostream &s) const**
  - write a responseRep object to an std::ostream (annotated format)

- **void read_tabular (std::istream &s)**
  - read functionValues from an std::istream (tabular format)

- **void write_tabular (std::ostream &s) const**
  - write functionValues to an std::ostream (tabular format)

- **void read (BiStream &s)**
  - read a responseRep object from a binary stream

- **void write (BoStream &s) const**
  - write a responseRep object to a binary stream

- **void read (MPIUnpackBuffer &s)**
  - read a responseRep object from a packed MPI buffer
void write (MPIPackBuffer &s) const  
write a responseRep object to a packed MPI buffer

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 RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set)  
update this response object from components of another response object

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)  
partially update this response object partial components of another response object

void reshape (size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag)  
rehapes response data arrays

void reset ()  
resets all response data to zero

void reset_inactive ()  
resets all inactive response data to zero

void active_set_request_vector (const ShortArray &asrv)  
set the active set request vector and verify consistent number of response functions

void active_set_derivative_vector (const SizetArray &asdv)  
set the active set derivative vector and reshape functionGradients/functionHessians if needed

Private Attributes

- int referenceCount  
  number of handle objects sharing responseRep
• RealVector functionValues
  abstract set of response functions

• RealMatrix functionGradients
  first derivatives of the response functions

• RealSymMatrixArray functionHessians
  second derivatives of the response functions

• ActiveSet responseActiveSet
  copy of the ActiveSet used by the Model to generate a Response instance

• StringArray functionLabels
  response function identifiers used to improve output readability

• String responsesId
  response identifier string from the input file

Friends

• class Response
  the handle class can access attributes of the body class directly

• bool operator==(const ResponseRep &rep1, const ResponseRep &rep2)
  equality operator

13.127.1 Detailed Description

Container class for response functions and their derivatives. ResponseRep provides the body class. The ResponseRep 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 (ResponseRep) actually contains the response data (functionValues, functionGradients, functionHessians, etc.). The representation is hidden in that an instance of ResponseRep 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).

13.127.2 Constructor & Destructor Documentation

13.127.2.1 ResponseRep (const Variables & vars, const ProblemDescDB & problem_db) [private]

standard constructor built from problem description database The standard constructor used by Dakota::ModelRep.
13.127 ResponseRep Class Reference

References
Variables::continuous_variable_ids(), Variables::cv(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues, ProblemDescDB::get_size(), ProblemDescDB::get_string(), ActiveSet::request_vector(), and ResponseRep::responseActiveSet.

13.127.2 ResponseRep (const ActiveSet & set) [private]

alternate constructor using limited data Used for building a response object of the correct size on the fly (e.g., by slave analysis servers performing execute() on a local_response). functionLabels is not needed for this purpose since it’s not passed in the MPI send/receive buffers. However, NPSOLOptimizer’s user-defined functions option uses this constructor to build bestResponseArray.front() and bestResponseArray.front() needs functionLabels for I/O, so construction of functionLabels has been added.

References Dakota::build_labels(), ResponseRep::functionGradients, ResponseRep::functionHessians, and ResponseRep::functionLabels.

13.127.3 Member Function Documentation

13.127.3.1 void read (std::istream & s) [private]

read a responseRep object from an std::istream ASCII version of read needs capabilities for capturing data omissions or formatting errors (resulting from user error or asynch race condition) and analysis failures (resulting from nonconvergence, instability, etc.).

References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues, Dakota::read_match(), Dakota::read_col_vector_trans(), ResponseRep::read_data(), ActiveSet::request_vector(), ResponseRep::reset(), and ResponseRep::responseActiveSet.

Referenced by Response::read().

13.127.3.2 void write (std::ostream & s) const [private]

write a responseRep object to an std::ostream ASCII version of write.

References Dakota::abort_handler(), Dakota::array_write_annotated(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::responseActiveSet, Dakota::write_col_vector_trans(), ResponseRep::write_data(), and Dakota::write_precision.

Referred by Response::write().

13.127.3.3 void read_annotated (std::istream & s) [private]

read a responseRep object from an std::istream (annotated format) read_annotated() is used for neutral file translation of restart files. Since objects are built solely from this data, annotations are used. This version closely mirrors the BiStream version.

References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, Dakota::read_col_vector_trans(), Dakota::read_lower_triangle(), Dakota::read_string().

Dakota Version 5.4 Developers Manual generated on November 7, 2013
ActiveSet::request_vector(), ResponseRep::reset(), ResponseRep::reshape(), ActiveSet::reshape(), and ResponseRep::responseActiveSet.

Referenced by Response::read_annotated().

13.127.3.4 void write_annotated (std::ostream & s) const  [private]

write a responseRep object to an std::ostream (annotated format) write_annotated() is used for neutral file translation of restart files. Since objects need to be build solely from this data, annotations are used. This version closely mirrors the BoStream version, with the exception of the use of white space between fields.

References Dakota::array_write_annotated(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::responseActiveSet, ActiveSet::write_annotated(), Dakota::write_col_vector_trans(), and Dakota::write_lower_triangle().

Referenced by Response::write_annotated().

13.127.3.5 void read_tabular (std::istream & s)  [private]

read functionValues from an std::istream (tabular format) read_tabular is used to read functionValues in tabular format. It is currently only used by ApproximationInterfaces in reading samples from a file. There is insufficient data in a tabular file to build complete response objects; rather, the response object must be constructed a priori and then its functionValues can be set.

References ResponseRep::functionValues.

Referenced by Response::read_tabular().

13.127.3.6 void write_tabular (std::ostream & s) const  [private]

write functionValues to an std::ostream (tabular format) write_tabular is used for output of functionValues in a tabular format for convenience in post-processing/plotting of DAKOTA results.

References ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::responseActiveSet, and Dakota::write_precision.

Referenced by Response::write_tabular().

13.127.3.7 void read (BiStream & s)  [private]

read a responseRep object from a binary stream Binary version differs from ASCII version in 2 primary ways: (1) it lacks formatting. (2) the Response has not been sized a priori. In reading data from the binary restart file, a ParamResponsePair was constructed with its default constructor which called the Response default constructor. Therefore, we must first read sizing data and resize all of the arrays.

References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, Dakota::read_col_vector_trans(), Dakota::read_lower_triangle(), ActiveSet::request_vector(), ResponseRep::reset(), ResponseRep::reshape(), and ResponseRep::responseActiveSet.
13.127.3.8 void write (BoStream & s) const [private]

write a responseRep object to a binary stream Binary version differs from ASCII version in 2 primary ways: (1) It lacks formatting. (2) In reading data from the binary restart file, ParamResponsePairs are constructed with their default constructor which calls the Response default constructor. Therefore, we must first write sizing data so that ResponseRep::read(BoStream& s) can resize the arrays.

References ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::responseActiveSet, Dakota::write_col_vector_trans(), and Dakota::write_lower_triangle().

13.127.3.9 void read (MPIUnpackBuffer & s) [private]

read a responseRep object from a packed MPI buffer UnpackBuffer version differs from BiStream version in the omission of functionLabels. Master processor retains labels and interface ids and communicates asv and response data only with slaves.


13.127.3.10 void write (MPIPackBuffer & s) const [private]

write a responseRep object to a packed MPI buffer MPIPackBuffer version differs from BoStream version only in the omission of functionLabels. The master processor retains labels and ids and communicates asv and response data only with slaves.

References ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::responseActiveSet, Dakota::write_col_vector_trans(), and Dakota::write_lower_triangle().

13.127.3.11 void update (const RealVector & source_fn_vals, const RealMatrix & source_fn_grads, const RealSymMatrixArray & source_fn_hessians, const ActiveSet & source_set) [private]

update this response object from components of another response object Copy function values/gradients/Hessians data _only_. Prevents unwanted overwriting of responseActiveSet, functionLabels, etc. Also, care is taken to account for differences in derivative variable matrix sizing.

References Dakota::abort_handler(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::reset_inactive(), and ResponseRep::responseActiveSet.

Referenced by Response::update().
13.127.3.12 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) [private]

partially update this response object partial components of another response object Copy function values/gradients/Hessians data _only_. Prevents unwanted overwriting of responseActiveSet, functionLabels, etc. Also, care is taken to account for differences in derivative variable matrix sizing.

References Dakota::abort_handler(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, ActiveSet::request_vector(), ResponseRep::reset_inactive(), and ResponseRep::responseActiveSet.

Referenced by Response::update_partial().

13.127.3.13 void reshape (size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag) [private]

reheapes response data arrays Reshape functionValues, functionGradients, and functionHessians according to num_fns, num_params, grad_flag, and hess_flag.

References Dakota::build_labels(), ActiveSet::derivative_vector(), ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionLabels, ResponseRep::functionValues, ActiveSet::request_vector(), ActiveSet::reshape(), and ResponseRep::responseActiveSet.

Referenced by ResponseRep::active_set_derivative_vector(), ResponseRep::read(), ResponseRep::read_annotated(), and Response::reshape().

13.127.3.14 void reset () [private]

resets all response data to zero Reset all numerical response data (not labels, ids, or active set) to zero.

References ResponseRep::functionGradients, ResponseRep::functionHessians, and ResponseRep::functionValues.

Referenced by ResponseRep::read(), ResponseRep::read_annotated(), and Response::reset().

13.127.3.15 void reset_inactive () [private]

resets all inactive response data to zero Used to clear out any inactive data left over from previous evaluations.

References ResponseRep::functionGradients, ResponseRep::functionHessians, ResponseRep::functionValues, ActiveSet::request_vector(), and ResponseRep::responseActiveSet.

Referenced by Response::reset_inactive(), ResponseRep::update(), and ResponseRep::update_partial().

13.127.4 Member Data Documentation

13.127.4.1 RealMatrix functionGradients [private]

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).

The documentation for this class was generated from the following files:

- DakotaResponse.hpp
- DakotaResponse.cpp
13.128  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

- template<typename StoredType >
  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 >
  StoredType get_data (const StrStrSizet &iterator_id, const std::string &data_name) const

  return requested data by value in StoredType

- template<typename StoredType >
  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 >
  const StoredType * get_data_ptr (const StrStrSizet &iterator_id, const std::string &result_key) const

  return pointer to stored data entry

- template<typename StoredType >
  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 add_data (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>
  
  StoredType cast_data (const boost::any &dataholder) const
  
  cast the reference to the any data to the requested type

- template<typename StoredType>
  
  const StoredType * cast_data_ptr (const boost::any *dataholder) const
  
  cast the pointer to the 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

- 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

Private Attributes

- std::map< ResultsKeyType, ResultsValueType > iteratorData
  
  core data storage (map from key to value type)
13.128.1 Detailed Description

Class: **ResultsDBAny** Description: A map-based container to store DAKOTA Iterator results in underlying boost::any's, with optional metadata.

13.128.2 Member Function Documentation

13.128.2.1 `void array_insert (const StrStrSizet & iterator_id, const std::string & data_name, size_t index, const StoredType & sent_data) [inline]`

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().
Referenced by ResultsManager::array_insert().

13.128.2.2 `void add_data (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(), and ResultsManager::insert().

13.128.2.3 `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
13.129 **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`.

- `const StoredType & const_view () const`
  
  return a reference to the stored data, whether from core or file

**Private Member Functions**

- `ResultsEntry ()`
  
  default construction disallowed: data must be initialized from DB lookup if needed

**Private Attributes**

- `bool coreActive`
  
  whether the `ResultsManager` has an active in-core database

- `StoredType dbData`
  
  data retrieved from file database

- `const StoredType * dbDataPtr`
  
  non-const pointer to const data we don’t own in the core case

### 13.129.1 **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.

The documentation for this class was generated from the following file:
• ResultsManager.hpp
13.130 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 run_iterator call creates or increments this count for its string identifier.

Public Member Functions

- size_t increment_id (const std::string &method_name, const std::string &method_id)
  explicitly increment the iterator results ID, init to 1 if needed

- size_t get_id (const std::string &method_name, const std::string &method_id)
  get (possibly creating) a unique iterator results ID for the passed name

- size_t get_id (const std::string &method_name, const std::string &method_id) 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
13.130.1 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 run_iterator 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
13.131 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 &text_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.

- **void insert (const StrStrSizet &iterator_id, const std::string &data_name, StringMultiArrayConstView sma_labels, const MetaDataType metadata=MetaDataType())**
  
  insert StringMultiArrayConstView, e.g. variable labels

- **void insert (const StrStrSizet &iterator_id, const std::string &data_name, const StringArray &sa_labels, const MetaDataType metadata=MetaDataType())**
  
  insert StringArray, e.g., response labels

- **void insert (const StrStrSizet &iterator_id, const std::string &data_name, const RealMatrix &matrix, const MetaDataType metadata=MetaDataType())**
  
  insert RealMatrix, e.g. correlations

- **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

Public Attributes

- **ResultsNames results_names**
  
  Copy of valid results names for when manager is passed around.
Private Member Functions

- \texttt{\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

- \texttt{\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)

- \texttt{\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

- \texttt{\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

- \texttt{\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

- \texttt{bool coreDBActive}
  whether the in-core database is active

- \texttt{std::string coreDBFilename}
  filename for the in-core database

- \texttt{bool fileDBActive}
  whether the file database is active

- \texttt{ResultsDBAny coreDB}
  In-core database, with option to flush to file at end.

Friends

- \texttt{class ResultsEntry}
  ResultsEntry is a friend of ResultsManager.
13.131 ResultsManager Class Reference

13.131.1 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. Probably want to use concrete types for arrays too.

All insertions overwrite any previous data.

The documentation for this class was generated from the following files:

- ResultsManager.hpp
- ResultsManager.cpp
13.132 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_drv
- 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 moment_cis
- std::string extreme_values
- std::string map_resp_prob
- std::string map_resp_rel
- 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 drv_labels
- std::string fn_labels
13.132 ResultsNames Class Reference

13.132.1 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
13.133 RichExtrapVerification Class Reference

Class for Richardson extrapolation for code and solution verification. Inheritance diagram for RichExtrapVerification:

Class for Richardson extrapolation for code and solution verification. Inheritance diagram for RichExtrapVerification:

```
RichExtrapVerification
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Analyzer</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Verification</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>RichExtrapVerification</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **RichExtrapVerification (Model &model)**
  
  *constructor*

- **~RichExtrapVerification ()**
  
  *destructor*

- **void perform_verification ()**
  
  _Redefines the run_iterator virtual function for the PStudy/DACE branch._

- **void print_results (std::ostream &s)**
  
  _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

- short studyType
  *internal code for extrapolation study type: ESTIMATE_ORDER, CONVERGE_ORDER, or CONVERGE_QOI*

- 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.*

13.133.1 Detailed Description

Class for Richardson extrapolation for code and solution verification. The RichExtrapVerification class contains several algorithms for performing Richardson extrapolation.

13.133.2 Member Function Documentation

13.133.2.1 void print_results (std::ostream & s) [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, RichExtrapVerification::refinementRate, RichExtrapVerification::refinementRefPt, Model::response_labels(), and Dakota::write_data().

### 13.133.2.2 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::perform_verification().

### 13.133.2.3 void converge_order () [private]

iterate using extrapolation() until convOrder stabilizes This algorithm continues to refine until the convergence order estimate converges.

References Iterator::convergenceTol, RichExtrapVerification::convOrder, Dakota::copy_data(), RichExtrapVerification::extrapolate_result(), RichExtrapVerification::extrapolation(), RichExtrapVerification::extrapQOI, RichExtrapVerification::factorIndex, RichExtrapVerification::initialCVars, Iterator::maxIterations, RichExtrapVerification::numErrorQOI, RichExtrapVerification::numFactors, Iterator::numFunctions, Iterator::outputLevel, RichExtrapVerification::refinementRate, RichExtrapVerification::refinementRefPt, and Dakota::write_data().

Referenced by RichExtrapVerification::perform_verification().

### 13.133.2.4 void converge_qoi () [private]

iterate using extrapolation() until QOIs stabilize This algorithm continues to refine until the discretization error lies within a prescribed tolerance.

References Iterator::convergenceTol, RichExtrapVerification::extrapolate_result(), RichExtrapVerification::extrapolation(), RichExtrapVerification::extrapQOI, RichExtrapVerification::factorIndex, RichExtrapVerification::initialCVars, Iterator::maxIterations, RichExtrapVerification::numErrorQOI, RichExtrapVerification::numFactors, Iterator::numFunctions, Iterator::outputLevel, RichExtrapVerification::refinementRate, RichExtrapVerification::refinementRefPt, and Dakota::write_data().

Referenced by RichExtrapVerification::perform_verification().

The documentation for this class was generated from the following files:

- RichExtrapVerification.hpp
- RichExtrapVerification.cpp
13.134 ScilabInterface Class Reference

Inheritance diagram for ScilabInterface:

```
          Interface
            |
            |
ApplicationInterface
            |
            |
DirectApplicInterface
            |
            |
          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*

13.134.1 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
13.135 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)
  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 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 drv_labels, const StringArray &resp_labels) const
  prints the correlations computed in compute_correlations()

Private Member Functions

- void simple_corr (RealMatrix &total_data, bool rank_on, const int &num_in)
  computes simple correlations

- void partial_corr (RealMatrix &total_data, bool rank_on, const int &num_in)
  computes partial correlations

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

13.135.1 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 PStudyDACE branch. Currently, the utility functions provide global sensitivity analysis through correlation calculations (e.g. simple, partial, rank, raw) as well as variance-based decomposition.

The documentation for this class was generated from the following files:

- SensAnalysisGlobal.hpp
- SensAnalysisGlobal.cpp
13.136 SequentialHybridStrategy Class Reference

**Strategy** for sequential hybrid minimization using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity. Inheritance diagram for SequentialHybridStrategy:

```
Strategy
   |
   v
HybridStrategy
   |
   v
SequentialHybridStrategy
```

**Public Member Functions**

- **SequentialHybridStrategy** (ProblemDescDB &problem_db)
  *constructor*

- **~SequentialHybridStrategy** ()
  *destructor*

**Protected Member Functions**

- **void run_strategy** ()
  *Performs the hybrid minimization strategy by executing multiple iterators on different models of varying fidelity.*

- **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)
  *initialize the iterator about to be executed within a parallel iterator scheduling function (serve_iterators() or static_schedule_iterators())*

- **void pack_parameters_buffer** (MPIPackBuffer &send_buffer, int job_index)
  *pack a send_buffer for assigning an iterator job to a server*

- **void unpack_parameters_buffer** (MPIUnpackBuffer &recv_buffer)
  *unpack a recv_buffer for accepting an iterator job from the scheduler*

- **void pack_results_buffer** (MPIPackBuffer &send_buffer, int job_index)
  *pack a send_buffer for returning iterator results from a server*
• void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  unpack a recv_buffer for accepting iterator results from a server

• void update_local_results (int job_index)
  update local PRP results arrays with current iteration results

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 (PRPArray &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_buffer(MPIUnpackBuffer) and initialize_iterator(int) to update the active Model and Iterator

Private Attributes

• String hybridType
  sequential or sequential_adaptive

• size_t seqCount
  hybrid sequence counter: 0 to numIterators-1

• Real progressMetric
  the amount of progress made in a single iterator++ cycle within a sequential adaptive hybrid

• 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

### 13.136.1 Detailed Description

**Strategy** for sequential hybrid minimization using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity. The sequential hybrid minimization strategy has 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 minimizer), and (2) the adaptive sequential hybrid uses adaptive stopping rules for the minimizers that are controlled externally by the strategy. Note that while the strategy is targeted at minimizers, any iterator may be used so long as it defines the notion of a final solution which can be passed as the starting point for subsequent iterators.

### 13.136.2 Member Function Documentation

#### 13.136.2.1 void pack_parameters_buffer (MPIPackBuffer & send_buffer, int job_index) [inline, protected, virtual]

pack a send_buffer for assigning an iterator job to a server This virtual function redefinition is executed on the dedicated master processor for self scheduling. It is not used for peer partitions.

Reimplemented from **Strategy**.

References SequentialHybridStrategy::extract_parameter_sets(), and SequentialHybridStrategy::seqCount.

Referenced by SequentialHybridStrategy::run_sequential().

#### 13.136.2.2 void unpack_parameters_buffer (MPIUnpackBuffer & recv_buffer) [inline, protected, virtual]

unpack a recv_buffer for accepting an iterator job from the scheduler This virtual function redefinition is executed on an iterator server for dedicated master self scheduling. It is not used for peer partitions.

Reimplemented from **Strategy**.

References SequentialHybridStrategy::initialize_iterator(), and SequentialHybridStrategy::seqCount.

#### 13.136.2.3 void pack_results_buffer (MPIPackBuffer & send_buffer, int job_index) [inline, protected, virtual]

pack a send_buffer for returning iterator results from a server This virtual function redefinition is executed either on an iterator server for dedicated master self scheduling or on peers 2 through n for static scheduling.

Reimplemented from **Strategy**.
References SequentialHybridStrategy::prpResults.

13.136.2.4 void unpack_results_buffer (MPIUnpackBuffer & recv_buffer, int job_index) [inline, protected, virtual]

unpack a recv_buffer for accepting iterator results from a server. This virtual function redefinition is executed on an strategy master (either the dedicated master processor for self scheduling or peer 1 for static scheduling).

Reimplemented from Strategy.

References SequentialHybridStrategy::prpResults.

13.136.2.5 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_i(), ParallelLibrary::bcast_si(), Response::function_values(), Strategy::graph2DFlag, Iterator::initialize_graphics(), Model::interface_id(), Response::is_null(), Variables::is_null(), Strategy::iteratorCommRank, Strategy::iteratorCommSize, Strategy::iteratorServerId, HybridStrategy::methodList, Iterator::num_final_solutions(), Strategy::numMasterJobs, HybridStrategy::numIterators, Strategy::numIteratorServers, SequentialHybridStrategy::pack_parameters_buffer(), Strategy::parallelLib, SequentialHybridStrategy::parameterSets, Strategy::paramsMsgLen, SequentialHybridStrategy::prpResults, ParallelLibrary::recv_si(), Iterator::response_results(), Strategy::resultsMsgLen, Iterator::returns_multiple_points(), Strategy::schedule_iterators(), HybridStrategy::selectedIterators, ParallelLibrary::send_si(), SequentialHybridStrategy::seqCount, MPIPackBuffer::size(), Strategy::stratIterDedMaster, Strategy::stratIterMessagePass, Strategy::tabularDataFile, Strategy::tabularDataFlag, HybridStrategy::userDefinedModels, Iterator::variables_results(), Strategy::worldRank, and Dakota::write_data().

Referenced by SequentialHybridStrategy::run_strategy().

13.136.2.6 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 Strategy::graph2DFlag, HybridStrategy::methodList, HybridStrategy::numIterators, SequentialHybridStrategy::progressMetric, SequentialHybridStrategy::progressThreshold, Strategy::run_iterator(), HybridStrategy::selectedIterators, SequentialHybridStrategy::seqCount, Strategy::tabularDataFile, Strategy::tabularDataFlag, HybridStrategy::userDefinedModels, and Strategy::worldRank.

Referenced by SequentialHybridStrategy::run_strategy().
13.136.2.7  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 strategy 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 SequentialHybridStrategy::parameterSets, and SequentialHybridStrategy::partition_sets().
Referenced by SequentialHybridStrategy::initialize_iterator(), and SequentialHybridStrategy::pack_parameters-_buffer().

The documentation for this class was generated from the following files:

- SequentialHybridStrategy.hpp
- SequentialHybridStrategy.cpp
13.137 SerialDirectApplicInterface Class Reference

Sample derived interface class for testing serial simulator plug-ins using assign_rep(). Inheritance diagram for SerialDirectApplicInterface::

```
<table>
<thead>
<tr>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationInterface</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
</tr>
<tr>
<td>SerialDirectApplicInterface</td>
</tr>
</tbody>
</table>
```

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_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_iterator_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.*

13.137.1 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 may be activated by specifying the `--with-plugin` configure option, which activates the DAKOTA_PLUGIN macro in dakota_config.h used by main.cpp (which activates the plug-in code block within that file) and activates the PLUGIN_S declaration defined in Makefile.include and used in Makefile.source (which add this class to the build). Test input files should then use an analysis_driver of "plugin_rosenbrock".

13.137.2 Member Function Documentation

13.137.2.1 void test_local_evaluations (Dakota::PRPQueue &prp_queue) [inline, protected, virtual]

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 Strategy::run_iterator() --> Model::serve() --> ApplicationInterface::serve_evaluations() --> ApplicationInterface::serve_evaluations_asynch()).

Reimplemented from DirectApplicInterface.

References SerialDirectApplicInterface::wait_local_evaluations().

The documentation for this class was generated from the following files:

- PluginSerialDirectApplicInterface.hpp
- PluginSerialDirectApplicInterface.cpp
13.138  **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 SizetArray &vars_comps_totals)**  
  *lightweight constructor*

- **SharedVariablesData (const SharedVariablesData &svd)**  
  *copy constructor*

- **~SharedVariablesData ()**  
  *destructor*

- **SharedVariablesData & operator= (const SharedVariablesData &svd)**  
  *assignment operator*

- **void size_all_continuous_labels (bool relax)**  
  *size labels for all of the continuous variables, with or without discrete relaxation*

- **void initialize_all_continuous_types (bool relax)**  
  *initialize types for all of the continuous variables, with or without discrete relaxation*

- **void initialize_all_continuous_ids (bool relax)**  
  *initialize ids for all of the continuous variables, with or without discrete relaxation*

- **void size_all_discrete_int_labels ()**  
  *size labels for all of the discrete integer variables*

- **void initialize_all_discrete_int_types ()**  
  *initialize types for all of the discrete integer variables*

- **void size_all_discrete_real_labels ()**  
  *size labels for all of the discrete real variables*

- **void initialize_all_discrete_real_types ()**  
  *initialize types for all of the discrete real variables*
• void initialize_active_components ()
  initialize the active components totals given active variable counts

• void initialize_inactive_components ()
  initialize the inactive components totals given inactive variable counts

• 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_real_labels (size_t start, size_t num_items) const
  get num_items discrete real labels beginning at index start

• void all_discrete_real_labels (StringMultiArrayConstView drv_labels, size_t start, size_t num_items)
  set num_items discrete real labels beginning at index start

• void all_discrete_real_label (const String &drv_label, size_t index)
  set discrete real 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

• UShortMultiArrayConstView all_discrete_int_types (size_t start, size_t num_items) const
  get num_items discrete integer types beginning at index start

• UShortMultiArrayConstView all_discrete_real_types (size_t start, size_t num_items) const
  get num_items discrete real types beginning at index start

• SizetMultiArrayConstView all_continuous_ids (size_t start, size_t num_items) const
  get num_items continuous ids beginning at index start

• const SizetArray & relaxed_discrete_ids () const
get ids of discrete variables that have been relaxed into continuous variable arrays

- 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 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 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 cv)`
  - *set start index of active continuous vars*

- `void div(size_t ndiv)`
  - *set number of active discrete int vars*

- `void div_start(size_t div)`
  - *set start index of active discrete int vars*

- `void drv(size_t ndrv)`
  - *set number of active discrete real vars*

- `void drv_start(size_t drv)`
  - *set start index of active discrete real vars*

- `void icv(size_t nicv)`
  - *set number of inactive continuous vars*

- `void icv_start(size_t icv)`
  - *set start index of inactive continuous vars*

- `void idiv(size_t nidiv)`
  - *set number of inactive discrete int vars*

- `void idiv_start(size_t idiv)`
  - *set start index of inactive disc int vars*
• void idrv (size_t nidrv)
  
  set number of inactive discrete real vars

• void idrv_start (size_t idrvs)
  
  set start index of inactive disc real vars

Private Attributes

• SharedVariablesDataRep * svdRep
  
  pointer to the body (handle-body idiom)

13.138.1 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). Shared-VariablesData 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.

The documentation for this class was generated from the following file:

• SharedVariablesData.hpp
13.139  SharedVariablesDataRep Class Reference

The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances.

Private Member Functions

- **SharedVariablesDataRep** (const ProblemDescDB &problem_db, const std::pair< short, short >&view)
  
  *standard constructor*

- **SharedVariablesDataRep** (const std::pair< short, short >&view, const SizetArray &vars_comps_totals)
  
  *lightweight constructor*

- **~SharedVariablesDataRep** ()
  
  *destructor*

- void **size_all_continuous_labels** (bool relax)
  
  *size allContinuousLabels, with or without discrete relaxation*

- void **initialize_all_continuous_types** (bool relax)
  
  *initialize allContinuousTypes, with or without discrete relaxation*

- void **initialize_all_continuous_ids** (bool relax)
  
  *initialize allContinuousIds, with or without discrete relaxation*

- void **size_all_discrete_int_labels** ()
  
  *size allDiscreteIntLabels*

- void **initialize_all_discrete_int_types** ()
  
  *initialize allDiscreteIntTypes*

- void **size_all_discrete_real_labels** ()
  
  *size allDiscreteRealLabels*

- void **initialize_all_discrete_real_types** ()
  
  *initialize allDiscreteRealTypes*

- void **initialize_active_components** ()
  
  *initialize activeVarsCompsTotals given {c,di,dr}vStart and num{C,DI,DR}V*

- void **initialize_inactive_components** ()
  
  *initialize inactiveVarsCompsTotals given {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*
Private Attributes

- String variablesId
  variables identifier string from the input file

- 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 real} {design,aleatory uncertain,epistemic uncertain,state}

- SizetArray activeVarsCompsTotals
  totals for active variable type counts for {continuous,discrete integer,discrete real} {design,aleatory uncertain,epistemic uncertain,state}

- SizetArray inactiveVarsCompsTotals
  totals for inactive variable type counts for {continuous,discrete integer,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 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 variables within allDiscreteIntVars

- 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 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 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 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 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

• SizetArray relaxedDiscreteIds
  array of discrete variable identifiers for which the discrete requirement is relaxed by merging them into a continuous array

• int referenceCount
  number of handle objects sharing svdRep

Friends

• class SharedVariablesData
13.139.1 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++).

13.139.2 Constructor & Destructor Documentation

13.139.2.1 `SharedVariablesDataRep (const ProblemDescDB & problem_db, const std::pair< short, short > & view) [private]

standard constructor This constructor is the one which must build the base class data for all derived classes. get_variables() instantiates a derived class handle and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_variables() again). Since the letter is the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Variables).

References `SharedVariablesDataRep::allContinuousLabels`, `SharedVariablesDataRep::allDiscreteIntLabels`, `SharedVariablesDataRep::allDiscreteRealLabels`, `Dakota::copy_data_partial()`, `ProblemDescDB::get_sizet()`, `ProblemDescDB::get_sizet()`, `SharedVariablesDataRep::initialize_all_continuous_ids()`, `SharedVariablesDataRep::initialize_all_continuous_types()`, `SharedVariablesDataRep::initialize_all_discrete_int_types()`, `SharedVariablesDataRep::initialize_all_discrete_real_types()`, `SharedVariablesDataRep::variablesComponents`, `SharedVariablesDataRep::variablesCompsTotals`, and `SharedVariablesDataRep::variablesView`.

13.139.3 Member Data Documentation

13.139.3.1 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.

Referenced by `SharedVariablesDataRep::initialize_all_continuous_ids()`.

The documentation for this class was generated from the following files:

- `SharedVariablesData.hpp`
- `SharedVariablesData.cpp`
13.140 SingleMethodStrategy Class Reference

Simple fall-through strategy for running a single iterator on a single model. Inheritance diagram for SingleMethodStrategy::

```
Strategy

SingleMethodStrategy
```

Public Member Functions

- **SingleMethodStrategy (ProblemDescDB &problem_db)**
  
  *constructor*

- **~SingleMethodStrategy ()**
  
  *destructor*

- **void run_strategy ()**
  
  *Perform the strategy by executing selectedIterator on userDefinedModel.*

- **const Variables & variables_results () const**
  
  *return the final solution from selectedIterator (variables)*

- **const Response & response_results () const**
  
  *return the final solution from selectedIterator (response)*

Private Attributes

- **Model userDefinedModel**
  
  *the model to be iterated*

- **Iterator selectedIterator**
  
  *the iterator*

13.140.1 Detailed Description

Simple fall-through strategy for running a single iterator on a single model. This strategy executes a single iterator on a single model. Since it does not provide coordination for multiple iterators and models, it can considered to be a "fall-through" strategy in that it allows control to fall through immediately to the iterator.

The documentation for this class was generated from the following files:
• SingleMethodStrategy.hpp
• SingleMethodStrategy.cpp
13.141  SingleModel Class Reference

Derived model class which utilizes a single interface to map variables into responses. Inheritance diagram for SingleModel:

```
    Model
     |   
    ---|---
       | SingleModel
```

Public Member Functions

- **SingleModel** (ProblemDescDB &problem_db)
  - constructor
- **~SingleModel** ()
  - destructor

Protected Member Functions

- **Interface & interface ()**
  - return userDefinedInterface
- void derived_compute_response (const ActiveSet &set)
  - portion of compute_response() specific to SingleModel (invokes a synchronous map() on userDefinedInterface)
- void derived_asynch_compute_response (const ActiveSet &set)
  - portion of asynch_compute_response() specific to SingleModel (invokes an asynchronous map() on userDefinedInterface)
- const IntResponseMap & derived_synchronize ()
  - portion of synchronize() specific to SingleModel (invokes synch() on userDefinedInterface)
- const IntResponseMap & derived_synchronize_nowait ()
  - portion of synchronize_nowait() specific to SingleModel (invokes synch_nowait() on userDefinedInterface)
- void component_parallel_mode (short mode)
  - SingleModel only supports parallelism in userDefinedInterface, so this virtual function redefinition is simply a sanity check.
- String 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)

• void derived_init_communicators (int max_iterator_concurrency, bool recurse_flag=true)
  
  set up SingleModel for parallel operations (request forwarded to userDefinedInterface)

• void derived_init_serial ()
  
  set up SingleModel for serial operations (request forwarded to userDefinedInterface).

• void derived_set_communicators (int max_iterator_concurrency, bool recurse_flag=true)
  
  set active parallel configuration for the SingleModel (request forwarded to userDefinedInterface)

• void derived_free_communicators (int max_iterator_concurrency, bool recurse_flag=true)
  
  deallocate communicator partitions for the SingleModel (request forwarded to userDefinedInterface)

• void serve (int max_iterator_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 SingleModel iteration is complete.

• const String & interface_id () const
  
  return the userDefinedInterface identifier

• int evaluation_id () const
  
  return the current evaluation id for the SingleModel (request forwarded to userDefinedInterface)

• bool evaluation_cache () const
  
  return flag indicated usage of an evaluation cache by the SingleModel (request forwarded to userDefinedInterface)

• void set_evaluation_reference ()
  
  set the evaluation counter reference points for the SingleModel (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 SingleModel (request forwarded to userDefinedInterface)

• virtual void eval_tag_prefix (const String &eval_id_str)
  
  set the hierarchical eval ID tag prefix
Private Attributes

- Interface userDefinedInterface
  
  the interface used for mapping variables to responses

13.141.1 Detailed Description

Derived model class which utilizes a single interface to map variables into responses. The SingleModel 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 a single interface to perform the function evaluations.

13.141.2 Member Function Documentation

13.141.2.1 void eval_tag_prefix (const String & eval_id_str) [protected, virtual]

set the hierarchical eval ID tag prefix SingleModel doesn’t need to change the tagging, so just forward to Interface
Reimplemented from Model.
References Interface::eval_tag_prefix(), and SingleModel::userDefinedInterface.
The documentation for this class was generated from the following files:

- SingleModel.hpp
- SingleModel.cpp
13.142  SNLLBase Class Reference

Base class for OPT++ optimization and least squares methods. Inheritance diagram for SNLLBase:

```
SNLLBase
   \|-- SNLLLeastSq
   \|-- SNLLOptimizer
```

Public Member Functions

- **SNLLBase ()**  
  default constructor

- **SNLLBase (Model &model)**  
  standard constructor

- **~SNLLBase ()**  
  destructor

Protected Member Functions

- **void copy_con_vals_dak_to_optpp (const RealVector &local_fn_vals, RealVector &g, const 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, const 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, const 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, const 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 (const int &num_cv, bool vendor_num_grad_flag, const String &finite_diff_type, const Real &fdss, const int &max_iter, const int &max_fn_evals, const Real &conv_tol, const
Real &grad_tol, const Real &max_step, bool bound_constr_flag, const int &num_constr, short output_lev, OPTPP::OptimizeClass *the_optimizer, 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 RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_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).

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

13.142.1 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
13.143 SNLLLeastSq Class Reference

Wrapper class for the OPT++ optimization library. Inheritance diagram for SNLLLeastSq::

- Iterator
- Minimizer
- LeastSq
- SNLLBase
- SNLLLeastSq

Public Member Functions

- **SNLLLeastSq (Model &model)**
  standard constructor

- **SNLLLeastSq (const String &method_name, Model &model)**
  alternate constructor for instantiations without ProblemDescDB support

- **~SNLLLeastSq ()**
  destructor

- **void minimize_residuals ()**
  Performs the iterations to determine the least squares solution.

Protected Member Functions

- **void initialize_run ()**
  invokes LeastSq::initialize_run(), SNLLBase::snll_initialize_run(), and performs other set-up

- **void post_run (std::ostream &s)**
  invokes snll_post_run and re-implements post_run (does not call parent) and performs other solution processing

- **void finalize_run ()**
  restores instances

Static Private Member Functions

- **static void nlf2_evaluator_gn (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, RealSymMatrix &hess_f, int &result_mode)**
objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation.

- static void constraint1_evaluator_gn (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)
  
  constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.

- static void constraint2_evaluator_gn (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, OPTPP::OptppArray<RealSymMatrix> &hess_g, int &result_mode)
  
  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.

Private Attributes

- **SNLLLeastSq * prevSnllLsqInstance**
  
  pointer to the previously active object instance used for restoration in the case of iterator/model recursion

- **OPTPP::NLP0 * nlfObjective**
  
  objective NLF base class pointer

- **OPTPP::NLP0 * nlfConstraint**
  
  constraint NLF base class pointer

- **OPTPP::NLP * nlpConstraint**
  
  constraint NLP pointer

- **OPTPP::NLF2 * nlf2**
  
  pointer to objective NLF for full Newton optimizers

- **OPTPP::NLF2 * nlf2Con**
  
  pointer to constraint NLF for full Newton optimizers

- **OPTPP::NLF1 * nlf1Con**
  
  pointer to constraint NLF for Quasi Newton optimizers

- **OPTPP::OptimizeClass * theOptimizer**
  
  optimizer base class pointer

- **OPTPP::OptNewton * optnewton**
  
  Newton optimizer pointer.

- **OPTPP::OptBCNewton * optbcnewton**
  
  Bound constrained Newton optimizer ptr.

- **OPTPP::OptDHNIPS * optdhnips**
  
  Disaggregated Hessian NIPS optimizer ptr.
13.143 SNLLLeastSq Class Reference

Static Private Attributes

- static SNLLLeastSq * snllLSqInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

13.143.1 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.

13.143.2 Member Function Documentation

13.143.2.1 void post_run (std::ostream & s) [protected, virtual]

invokes snll_post_run and re-implements post_run (does not call parent) and performs other solution processing. SNLLLeastSq requires fn DB lookup, so overrides LeastSq::post_run and directly invokes Iterator::post_run when complete.

Reimplemented from LeastSq.

References Iterator::activeSet, Iterator::bestResponseArray, Iterator::bestVariablesArray, SNLLBase::copy_con_vals_optpp_to_dak(), Dakota::copy_data_partial(), Minimizer::cvScaleMultipliers, Minimizer::cvScaleTypes, Dakota::data_pairs, Minimizer::expData, Response::function_values(), LeastSq::get_confidence_intervals(), Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Minimizer::modify_s2n(), Minimizer::need_resp_trans_byvars, SNLLLeastSq::nlfObjective, Minimizer::numExperiments, Iterator::numFunctions, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Minimizer::numReplicates, Minimizer::numUserPrimaryFns, Minimizer::obsDataFlag, ActiveSet::request_values(), ActiveSet::request_vector(), Minimizer::responseScaleMultipliers, Minimizer::responseScaleOffsets, Minimizer::responseScaleTypes, Minimizer::secondaryRespScaleFlag, SNLLBase::snll_post_run(), SNLLLeastSq::theOptimizer, and Minimizer::varsScaleFlag.
13.143.2.2 void nlf2_evaluator_gn (int mode, int n, const RealVector & x, double & f, RealVector & grad_f, RealSymMatrix & hess_f, int & result_mode) [static, private]

objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation. This nlf2 evaluator function is used for the Gauss-Newton method in order to exploit the special structure of the nonlinear least squares problem. Here, \( f(x) = \sum (T_i - Tbar_i)^2 \) and Response is made up of residual functions and their gradients along with any nonlinear constraints. The objective function and its gradient vector and Hessian matrix are computed directly from the residual functions and their derivatives (which are returned from the Response object).

References Dakota::abort_handler(), Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Iterator::numFunctions, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Iterator::outputLevel, ActiveSet::request_vector(), SNLLLeastSq::snllLSqInstance, Dakota::write_data(), and Dakota::write_precision.

Referenced by SNLLLeastSq::SNLLLeastSq().

13.143.2.3 void constraint1_evaluator_gn (int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, int & result_mode) [static, private]

constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods. While it does not employ the Gauss-Newton approximation, it is distinct from constraint1_evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with diaggregated Hessian NIPS and is currently active.

References Dakota::abort_handler(), Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Iterator::numFunctions, LeastSq::numLeastSqTerms, Iterator::outputLevel, ActiveSet::request_vector(), and SNLLLeastSq::snllLSqInstance.

Referenced by SNLLLeastSq::SNLLLeastSq().

13.143.2.4 void constraint2_evaluator_gn (int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, OPTPP::OptppArray< RealSymMatrix > & hess_g, int & result_mode) [static, private]

constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods. While it does not employ the Gauss-Newton approximation, it is distinct from constraint2_evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with full Newton NIPS and is currently inactive.

References Dakota::abort_handler(), Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_hess(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, SNLLBase::modeOverrideFlag, Iterator::numFunctions,
LeastSq::numLeastSqTerms, Iterator::outputLevel, ActiveSet::request_vector(), and SNLLLeastSq::snllLsqInstance.

The documentation for this class was generated from the following files:

- SNLLLeastSq.hpp
- SNLLLeastSq.cpp
13.144 SNLLOptimizer Class Reference

Wrapper class for the OPT++ optimization library. Inheritance diagram for SNLLOptimizer::

```
  SNLLOptimizer
   \|/
  Minimizer
   |\|
  Optimizer
   | \|
  SNLLBase
```

Public Member Functions

- **SNLLOptimizer (Model &model)**
  - standard constructor

- **SNLLOptimizer (const String &method_name, Model &model)**
  - alternate constructor for instantiations "on the fly"

- **SNLLOptimizer (const RealVector &initial_pt, const RealVector &var_l_bnds, const RealVector &var_u_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_tgts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, void(*user_obj_eval)(int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode), void(*user_con_eval)(int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode))**
  - alternate constructor for instantiations "on the fly"

- **~SNLLOptimizer ()**
  - destructor

- **void find_optimum ()**
  - Performs the iterations to determine the optimal solution.

Protected Member Functions

- **void initialize_run ()**
  - invokes Optimizer::initialize_run(), SNLLBase::snll_initialize_run(), and performs other set-up

- **void post_run (std::ostream &s)**
  - performs data recovery and calls Optimizer::post_run()
• void finalize_run()
  performs cleanup, restores instances and calls parent finalize

Static Private Member Functions

• static void nlf0_evaluator (int n, const RealVector &x, double &f, int &result_mode)
  objective function evaluator function for OPT++ methods which require only function values.

• static void nlf1_evaluator (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode)
  objective function evaluator function which provides function values and gradients to OPT++ methods.

• static void nlf2_evaluator (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, RealSymMatrix &hess_f, int &result_mode)
  objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

• static void constraint0_evaluator (int n, const RealVector &x, RealVector &g, int &result_mode)
  constraint evaluator function for OPT++ methods which require only constraint values.

• static void constraint1_evaluator (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)
  constraint evaluator function which provides constraint values and gradients to OPT++ methods.

• static void constraint2_evaluator (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, OPTPP::OptppArray<RealSymMatrix> &hess_g, int &result_mode)
  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.

Private Attributes

• SNLLOptimizer * prevSnllOptInstance
  pointer to the previously active object instance used for restoration in the case of iterator/model recursion

• OPTPP::NLP0 * nlfObjective
  objective NLF base class pointer

• OPTPP::NLP0 * nlfConstraint
  constraint NLF base class pointer

• OPTPP::NLP * nlpConstraint
  constraint NLP pointer

• OPTPP::NLF0 * nlf0
pointer to objective NLF for nongradient optimizers

- OPTPP::NLF1 * nlf1
  pointer to objective NLF for (analytic) gradient-based optimizers

- OPTPP::NLF1 * nlf1Con
  pointer to constraint NLF for (analytic) gradient-based optimizers

- OPTPP::FDNLF1 * fdnlf1
  pointer to objective NLF for (finite diff) gradient-based optimizers

- OPTPP::FDNLF1 * fdnlf1Con
  pointer to constraint NLF for (finite diff) gradient-based optimizers

- OPTPP::NLF2 * nlf2
  pointer to objective NLF for full Newton optimizers

- OPTPP::NLF2 * nlf2Con
  pointer to constraint NLF for full Newton optimizers

- OPTPP::OptimizeClass * theOptimizer
  optimizer base class pointer

- OPTPP::OptPDS * optpds
  PDS optimizer pointer.

- OPTPP::OptCG * optcg
  CG optimizer pointer.

- OPTPP::OptLBFGS * optlbfgs
  L-BFGS optimizer pointer.

- OPTPP::OptNewton * optnewton
  Newton optimizer pointer.

- OPTPP::OptQNewton * optqnewton
  Quasi-Newton optimizer pointer.

- OPTPP::OptFDNewton * optfdnewton
  Finite Difference Newton opt pointer.

- OPTPP::OptBCNewton * optbcnewton
  Bound constrained Newton opt pointer.

- OPTPP::OptBCQNewton * optbcqnewton
  Bnd constrained Quasi-Newton opt ptr.
• OPTPP::OptBCFDNewton * optbcfdnewton
  
  Bnd constrained FD-Newton opt ptr.

• OPTPP::OptNIPS * optnips
  
  NIPS optimizer pointer.

• OPTPP::OptQNIPS * optqnips
  
  Quasi-Newton NIPS optimizer pointer.

• OPTPP::OptFDNIPS * optfdnips
  
  Finite Difference NIPS opt pointer.

• String setUpType
  
  flag for iteration mode: "model" (normal usage) or "user_functions" (user-supplied functions mode for "on the fly" instantiations). NonDReliability currently uses the user_functions mode.

• RealVector initialPoint
  
  holds initial point passed in for "user_functions" mode.

• RealVector lowerBounds
  
  holds variable lower bounds passed in for "user_functions" mode.

• RealVector upperBounds
  
  holds variable upper bounds passed in for "user_functions" mode.

Static Private Attributes

• static SNLLOptimizer * snllOptInstance
  
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

13.144.1 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 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.

13.144.2 Constructor & Destructor Documentation

13.144.2.1 SNLLOptimizer (Model & model)

standard constructor This constructor is used for normal instantiations using data from the ProblemDescDB.

References Dakota::abort_handler(), Minimizer::boundConstraintFlag, SNLLBase::centeringParam, SNLLOptimizer::constraint0_evaluator(), SNLLOptimizer::constraint1_evaluator(), SNLLOptimizer::constraint2_evaluator(), Iterator::convergenceTol, Iterator::fdGradStepSize, SNLLOptimizer::fdnlf1, SNLLOptimizer::fdnlf1Con, ProblemDescDB::get_int(), ProblemDescDB::get_realm(), Model::init_communicators(), SNLLBase::init_fn(), Iterator::intervalType, Iterator::iteratedModel, Dakota::LARGE_SCALE, Iterator::maxConcurrency, Iterator::maxFunctionEvals, Iterator::maxIterations, SNLLBase::maxStep, SNLLBase::meritFn, Iterator::methodName, Minimizer::minimizerRecasts, SNLLOptimizer::nlf0, SNLLOptimizer::nlf0_evaluator(), SNLLOptimizer::nlf1, SNLLOptimizer::nlf1_evaluator(), SNLLOptimizer::nlf1Con, SNLLOptimizer::nlf2, SNLLOptimizer::nlf2_evaluator(), SNLLOptimizer::nlf2Con, SNLLOptimizer::nlfConstraint, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpConstraint, Minimizer::numConstraints, Iterator::numContinuousVars, Minimizer::numNonlinearConstraints, SNLLOptimizer::optbcqnewton, SNLLOptimizer::optbfgs, SNLLOptimizer::optbqnips, SNLLOptimizer::optcg, SNLLOptimizer::optfdnewton, SNLLOptimizer::optfdnips, SNLLOptimizer::optlbfgs, SNLLOptimizer::optlqnewton, SNLLOptimizer::optnips, SNLLOptimizer::optpds, SNLLOptimizer::optpqnips, SNLLOptimizer::optqnewton, SNLLOptimizer::optqnips, Iterator::outputLevel, SNLLOptimizer::theOptimizer, and Minimizer::vendorNumericalGradFlag.

13.144.2.2 SNLLOptimizer (const String & method_name, Model & model)

alternate constructor for instantiations "on the fly" This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

References Minimizer::boundConstraintFlag, SNLLOptimizer::constraint1_evaluator(), Iterator::convergenceTol, Iterator::fdGradStepSize, SNLLBase::init_fn(), Iterator::intervalType, Dakota::LARGE_SCALE, Iterator::maxFunctionEvals, Iterator::maxIterations, SNLLBase::meritFn, Iterator::methodName, SNLLOptimizer::nlf1, SNLLOptimizer::nlf1_evaluator(), SNLLOptimizer::nlf1Con, SNLLOptimizer::nlfConstraint, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpConstraint, Minimizer::numConstraints, Iterator::numContinuousVars, Minimizer::numNonlinearConstraints, SNLLOptimizer::optbcqnewton, SNLLOptimizer::optbfgs, SNLLOptimizer::optbqnips, SNLLOptimizer::optcg, SNLLOptimizer::optfdnewton, SNLLOptimizer::optfdnips, SNLLOptimizer::optlbfgs, SNLLOptimizer::optlqnewton, SNLLOptimizer::optnips, SNLLOptimizer::optpds, SNLLOptimizer::optpqnips, SNLLOptimizer::optqnewton, SNLLOptimizer::optqnips, Iterator::outputLevel, SNLLOptimizer::theOptimizer, and Minimizer::vendorNumericalGradFlag.
13.144.2.3 **SNLLOptimizer** (const RealVector & initial_pt, const RealVector & var_l_bnds, const RealVector & var_u_bnds, const RealMatrix & lin_ineq_coeffs, const RealVector & lin_ineq_l_bnds, const RealVector & lin_ineq_u_bnds, const RealMatrix & lin_eq_coeffs, const RealVector & lin_eq_tgts, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealMatrix & lin_eq_coeffs, const RealVector & nln_eq_tgts, const RealVector & nln_eq_tgts, void(*)(int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode) user_obj_eval, void(*)(int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode) user_con_eval)

This is an alternate constructor for instantiations "on the fly" This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers.

References Minimizer::bigRealBoundSize, Minimizer::boundConstraintFlag, SNLLBase::init_fn(), SNLLOptimizer::initialPoint, Dakota::LARGE_SCALE, SNLLOptimizer::lowerBounds, SNLLBase::meritFn, SNLLOptimizer::nlf1, SNLLOptimizer::nlf1Con, SNLLOptimizer::nlfConstraint, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpConstraint, Minimizer::numConstraints, Iterator::numContinuousVars, Minimizer::numNonlinearConstraints, SNLLOptimizer::optbcqnewton, SNLLOptimizer::optbfgs, SNLLOptimizer::optqnewton, SNLLOptimizer::optqps, Iterator::outputLevel, SNLLBase::searchStrat, SNLLBase::snll_initialize_run(), SNLLBase::snll_post_instantiate(), SNLLBase::snll_pre_instantiate(), SNLLOptimizer::theOptimizer, and SNLLOptimizer::upperBounds.

13.144.3 **Member Function Documentation**

### 13.144.3.1 void nlf0_evaluator (int n, const RealVector & x, double & f, int & result_mode) [static, private]

Objective function evaluator function for OPT++ methods which require only function values. For use when DAKOTA computes f and gradients are not directly available. This is used by nongradient-based optimizers such as PDS and by gradient-based optimizers in vendor numerical gradient mode (opt++'s internal finite difference routine is used).

References Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_value(), Iterator::iteratedModel, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numNonlinearConstraints, Iterator::outputLevel, Model::primary_response_fn_sense(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

### 13.144.3.2 void nlf1_evaluator (int mode, int n, const RealVector & x, double & f, RealVector & grad_f, int & result_mode) [static, private]

Objective function evaluator function which provides function values and gradients to OPT++ methods. For use when DAKOTA computes f and df/dX (regardless of gradientType). Vendor numerical gradient case is handled by nlf0_evaluator.

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradient_copy(), Response::function_value(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numNonlinearConstraints, Iterator::outputLevel, Model::primary_response_fn_sense(), ActiveSet::request_values(), and SNLLOptimizer::snllOptInstance.
objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods. For use when DAKOTA receives f, df/dX, & d^2f/dx^2 from the ApplicationInterface (analytic only). Finite differencing does not make sense for a full Newton approach, since lack of analytic gradients & Hessian should dictate the use of quasi-newton or fd-newton. Thus, there is no fdnlf2_evaluator for use with full Newton approaches, since it is preferable to use quasi-newton or fd-newton with nlfi1. Gauss-Newton does not fit this model; it uses nlfi2_evaluator instead of nlf2_evaluator.

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradient_copy(), Response::function_hessian(), Response::function_value(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numNonlinearConstraints, Iterator::outputLevel, Model::primary_response_fn_sense(), ActiveSet::request_values(), and SNLLOptimizer::nlsoptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

constraint evaluator function for OPT++ methods which require only constraint values. For use when DAKOTA computes g and gradients are not directly available. This is used by nongradient-based optimizers and by gradient-based optimizers in vendor numerical gradient mode (opt++'s internal finite difference routine is used).

References Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Optimizer::numObjectiveFns, Iterator::outputLevel, and SNLLOptimizer::nlsoptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

constraint evaluator function which provides constraint values and gradients to OPT++ methods. For use when DAKOTA computes g and dg/dX (regardless of gradientType). Vendor numerical gradient case is handled by constraint0_evaluator.

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_vars_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Optimizer::numObjectiveFns, Iterator::outputLevel, ActiveSet::request_values(), and SNLLOptimizer::nlsoptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().
13.144.3.6  void constraint2_evaluator (int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, OPTPP::OptppArray< RealSymMatrix > & hess_g, int & result_mode) [static, private]

constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods. For use when DAKOTA computes $g$, $dg/dX$, & $d^2g/dx^2$ (analytic only).

References Iterator::activeSet, Model::compute_response(), Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_hess(), SNLLBase::copy_vals_dak_to_optpp(), Model::current_response(), Response::function_gradients(), Response::function_hessians(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Optimizer::numObjectiveFs, Iterator::outputLevel, ActiveSet::request_values(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

The documentation for this class was generated from the following files:

- SNLLOptimizer.hpp
- SNLLOptimizer.cpp
13.145 SOLBase Class Reference

Base class for Stanford SOL software. Inheritance diagram for SOLBase::

```
SOLBase
  ↓
NLSSOLLeastSq  NPSOLOptimizer
```

### Public Member Functions
- **SOLBase ()**
  *default constructor*
- **SOLBase (Model &model)**
  *standard constructor*
- **~SOLBase ()**
  *destructor*

### Protected Member Functions
- **void allocate_arrays (const int &num_cv, const 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 (const int &num_cv, const int &num_nln_con, const int &num_lin_con, const 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, const int &verify_lev, const Real &fn_prec, const Real &linesrch_tol, const int &max_iter, const Real &constr_tol, const Real &conv_tol, const std::string &grad_type, const Real &fdss)**
  *Sets SOL method options using calls to npoptn2.*
- **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 *jac, 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 fnEvalCntr
  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

### 13.145.1 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
13.146 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.

13.146.1 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
13.147 Strategy Class Reference

Base class for the strategy class hierarchy. Inheritance diagram for Strategy:

```
Strategy
   |   |   |
  +---+---+---+
  |   |   |   |
  +---+---+---+
ConcurrentStrategy  HybridStrategy  SingleMethodStrategy
  |   |   |   |
  +---+---+---+
CollaborativeHybridStrategy  EmbeddedHybridStrategy  SequentialHybridStrategy
```

Public Member Functions

- **Strategy ()**
  
  *default constructor*

- **Strategy (ProblemDescDB &problem_db)**
  
  *envelope constructor*

- **Strategy (const Strategy &strat)**
  
  *copy constructor*

- virtual **~Strategy ()**
  
  *destructor*

- **Strategy operator= (const Strategy &strat)**
  
  *assignment operator*

- virtual **void run_strategy ()**
  
  *the run function for the strategy: invoke the iterator(s) on the model(s). Called from main.cpp.*

- virtual **const Variables & variables_results () const**
  
  *return the final strategy solution (variables)*

- virtual **const Response & response_results () const**
  
  *return the final strategy solution (response)*

- **ProblemDescDB & problem_description_db () const**
  
  *returns the problem description database (probDescDB)*
Protected Member Functions

- **Strategy** *(BaseConstructor, 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)*

- virtual void **initialize_iterator** *(int index)*
  
  - initialize the iterator about to be executed within a parallel iterator scheduling function *(serve_iterators() or static_schedule_iterators())*

- virtual void **pack_parameters_buffer** *(MPIPackBuffer &send_buffer, int job_index)*
  
  - pack a send_buffer for assigning an iterator job to a server

- virtual void **unpack_parameters_buffer** *(MPIUnpackBuffer &recv_buffer)*
  
  - unpack a recv_buffer for accepting an iterator job from the scheduler

- virtual void **pack_results_buffer** *(MPIPackBuffer &send_buffer, int job_index)*
  
  - pack a send_buffer for returning iterator results from a server

- virtual void **unpack_results_buffer** *(MPIUnpackBuffer &recv_buffer, int job_index)*
  
  - unpack a recv_buffer for accepting iterator results from a server

- virtual void **update_local_results** *(int job_index)*
  
  - update local PRP results arrays with current iteration results

- void **init_iterator_parallelism** ()
  
  - convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart.

- void **init_iterator** *(Iterator &the_iterator, Model &the_model)*
  
  - convenience function for allocating comms prior to running an iterator

- void **run_iterator** *(Iterator &the_iterator, Model &the_model)*
  
  - convenience function for invoking an iterator and managing parallelism. This version omits communicator repartitioning. Function must be public due to use by MINLPNode.

- void **free_iterator** *(Iterator &the_iterator, Model &the_model)*
  
  - convenience function for deallocating comms after running an iterator

- void **schedule_iterators** *(Iterator &the_iterator, Model &the_model)*
  
  - short convenience function for distributing control among self_schedule_iterators(), serve_iterators(), and static_schedule_iterators() (called by derived run_strategy())

- void **self_schedule_iterators** *(Model &the_model)*
  
  - executed by the strategy master to self-schedule iterator jobs among slave iterator servers
void serve_iterators (Iterator &the_iterator, Model &the_model)
executed on the slave iterator servers to perform iterator jobs assigned by the strategy master (called by derived run_strategy())

void static_schedule_iterators (Iterator &the_iterator, Model &the_model)
executed on iterator peers to statically schedule iterator jobs (called by derived run_strategy())

Protected Attributes

- ProblemDescDB & probDescDB
  class member reference to the problem description database

- ParallelLibrary & parallelLib
  class member reference to the parallel library

- String strategyName
  type of strategy: single_method, hybrid, multi_start, or pareto_set.

- bool stratIterMessagePass
  flag for message passing at si level

- bool stratIterDedMaster
  flag for dedicated master part. at si level

- int worldRank
  processor rank in MPI_COMM_WORLD

- int worldSize
  size of MPI_COMM_WORLD

- int iteratorCommRank
  processor rank in iteratorComm

- int iteratorCommSize
  number of processors in iteratorComm

- int numIteratorServers
  number of concurrent iterator partitions

- int iteratorServerId
  identifier for an iterator server

- bool graph2DFlag
  flag for using 2D graphics plots
• bool tabularDataFlag
  flag for file tabulation of graphics data

• String tabularDataFile
  filename for tabulation of graphics data

• bool resultsOutputFlag
  whether to output results data

• std::string resultsOutputFile
  filename for results data

• int maxConcurrency
  maximum iterator concurrency possible in Strategy

• int numIteratorJobs
  number of iterator executions to schedule

• int paramsMsgLen
  length of MPI buffer for parameter input instance(s)

• int resultsMsgLen
  length of MPI buffer for results output instance(s)

Private Member Functions

• Strategy * get_strategy ()
  Used by the envelope to instantiate the correct letter class.

Private Attributes

• Strategy * strategyRep
  pointer to the letter (initialized only for the envelope)

• int referenceCount
  number of objects sharing strategyRep

13.147.1 Detailed Description

Base class for the strategy class hierarchy. The Strategy class is the base class for the class hierarchy providing the top level control in DAKOTA. The strategy is responsible for creating and managing iterators and models. For memory efficiency and enhanced polymorphism, the strategy hierarchy employs the "letter/envelope idiom"
(see Coplien "Advanced C++", p. 133), for which the base class (Strategy) serves as the envelope and one of the derived classes (selected in Strategy::get_strategy()) serves as the letter.

13.147.2 Constructor & Destructor Documentation

13.147.2.1 Strategy ()

default constructor Default constructor. strategyRep is NULL in this case (a populated problem_db is needed to build a meaningful Strategy object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

13.147.2.2 Strategy (ProblemDescDB & problem_db)

envelope constructor Used in main.cpp instantiation to build the envelope. This constructor only needs to extract enough data to properly execute get_strategy, since Strategy::Strategy(BaseConstructor, problem_db) builds the actual base class data inherited by the derived strategies.

References Dakota::abort_handler(), Strategy::get_strategy(), and Strategy::strategyRep.

13.147.2.3 Strategy (const Strategy & strat)

copy constructor Copy constructor manages sharing of strategyRep and incrementing of referenceCount.


13.147.2.4 ~Strategy () [virtual]

destructor Destructor decrements referenceCount and only deletes strategyRep when referenceCount reaches zero.

References ParallelLibrary::free_iterator_communicators(), ParallelLibrary::is_null(), Strategy::parallelLib, Strategy::referenceCount, and Strategy::strategyRep.

13.147.2.5 Strategy (BaseConstructor, 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 strategies. get_strategy() 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_strategy() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Strategy).

References ProblemDescDB::get_int(), Strategy::probDescDB, and Dakota::write_precision.
### 13.147.3 Member Function Documentation

#### 13.147.3.1 Strategy operator= (const Strategy & strat)


#### 13.147.3.2 void pack_parameters_buffer (MPIPackBuffer & send_buffer, int job_index)

[protected, virtual]

pack a send_buffer for assigning an iterator job to a server This virtual function redefinition is executed on the dedicated master processor for self scheduling. It is not used for peer partitions.

Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

References Strategy::pack_parameters_buffer(), and Strategy::strategyRep.

Referenced by Strategy::pack_parameters_buffer(), and Strategy::self_schedule_iterators().

#### 13.147.3.3 void unpack_parameters_buffer (MPIUnpackBuffer & recv_buffer)

[protected, virtual]

unpack a recv_buffer for accepting an iterator job from the scheduler This virtual function redefinition is executed on an iterator server for dedicated master self scheduling. It is not used for peer partitions.

Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

References Strategy::strategyRep, and Strategy::unpack_parameters_buffer().

Referenced by Strategy::serve_iterators(), and Strategy::unpack_parameters_buffer().

#### 13.147.3.4 void pack_results_buffer (MPIPackBuffer & send_buffer, int job_index)

[protected, virtual]

pack a send_buffer for returning iterator results from a server This virtual function redefinition is executed either on an iterator server for dedicated master self scheduling or on peers 2 through n for static scheduling.

Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

References Strategy::pack_results_buffer(), and Strategy::strategyRep.

Referenced by Strategy::pack_results_buffer(), Strategy::serve_iterators(), and Strategy::static_schedule_iterators().

#### 13.147.3.5 void unpack_results_buffer (MPIUnpackBuffer & recv_buffer, int job_index)

[protected, virtual]

unpack a recv_buffer for accepting iterator results from a server This virtual function redefinition is executed on an strategy master (either the dedicated master processor for self scheduling or peer 1 for static scheduling).
Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

References Strategy::strategyRep, and Strategy::unpack_results_buffer().

Referenced by Strategy::self_schedule_iterators(), Strategy::static_schedule_iterators(), and Strategy::unpack_results_buffer().

13.147.3.6 void init_iterator_parallelism () [protected]

convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart. This function is called from derived class constructors once maxConcurrency is defined but prior to instantiating Iterators and Models.

References ParallelLevel::dedicated_master_flag(), ProblemDescDB::get_int(), ProblemDescDB::get_string(), ParallelLibrary::init_iterator_communicators(), Strategy::iteratorCommRank, Strategy::iteratorCommSize, Strategy::iteratorServerId, ParallelLibrary::manage_outputs_restart(), Strategy::maxConcurrency, ParallelLevel::message_pass(), ParallelLevel::num_servers(), Strategy::numIteratorServers, Strategy::parallelLib, Strategy::probDescDB, Strategy::resultsOutputFile, Strategy::resultsOutputFlag, ParallelLevel::server_communicator_rank(), ParallelLevel::server_communicator_size(), ParallelLevel::server_id(), Strategy::stratIterDedMaster, and Strategy::stratIterMessagePass.

Referenced by CollaborativeHybridStrategy::CollaborativeHybridStrategy(), ConcurrentStrategy::ConcurrentStrategy(), EmbeddedHybridStrategy::EmbeddedHybridStrategy(), SequentialHybridStrategy::SequentialHybridStrategy(), and SingleMethodStrategy::SingleMethodStrategy().

13.147.3.7 void init_iterator (Iterator & the_iterator, Model & the_model) [protected]

convenience function for allocating comms prior to running an iterator. This is a convenience function for encapsulating the allocation of communicators prior to running an iterator. It does not require a strategyRep forward since it is only used by letter objects.

References ProblemDescDB::get_iterator(), Model::init_comms_bcast_flag(), Model::init_communicators(), Strategy::iteratorCommRank, Strategy::iteratorCommSize, Iterator::maximum_concurrency(), Strategy::probDescDB, Model::serve_configurations(), and Model::stop_configurations().

Referenced by HybridStrategy::allocate_methods(), ConcurrentStrategy::ConcurrentStrategy(), and SingleMethodStrategy::SingleMethodStrategy().

13.147.3.8 void run_iterator (Iterator & the_iterator, Model & the_model) [protected]

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 those strategies which involve multiple iterator executions but only require communicator allocation/deallocation to be performed once.

It does not require a strategyRep forward since it is only used by letter objects. While it is currently a public function due to its use in MINLPNode, this usage still involves a strategy letter object.

References Strategy::iteratorCommRank, Iterator::maximum_concurrency(), Iterator::run_iterator(), Model::serve(), and Model::stop_servers().
13.147.3.9 **void free_iterator (Iterator & the_iterator, Model & the_model)** [protected]

convenience function for deallocating comms after running an iterator This is a convenience function for encapsulating the deallocation of communicators after running an iterator. It does not require a strategyRep forward since it is only used by letter objects.

References Model::free_communicators(), and Iterator::maximum_concurrency().

Referenced by HybridStrategy::deallocate_methods(), ConcurrentStrategy::~ConcurrentStrategy(), and SingleMethodStrategy::~SingleMethodStrategy().

13.147.3.10 **void schedule_iterators (Iterator & the_iterator, Model & the_model)** [protected]

short convenience function for distributing control among self_schedule_iterators(), serve_iterators(), and 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.


Referenced by SequentialHybridStrategy::run_sequential(), EmbeddedHybridStrategy::run_strategy(), ConcurrentStrategy::run_strategy(), and CollaborativeHybridStrategy::run_strategy().

13.147.3.11 **void self_schedule_iterators (Model & the_model)** [protected]

executed by the strategy master to self-schedule iterator jobs among slave iterator servers (called by derived run_strategy()) This function is adapted from ApplicationInterface::self_schedule_evaluations(). It occurs on a dedicated master scheduler, so output is directed to std::cout.

References ParallelLibrary::free(), ParallelLibrary::irecv_si(), ParallelLibrary::isend_si(), Strategy::numIteratorJobs, Strategy::numIteratorServers, Strategy::pack_parameters_buffer(), Strategy::parallelLib, ParallelLibrary::print_configuration(), MPIPackBuffer::reset(), MPIUnpackBuffer::resize(), Strategy::resultsMsgLen, Strategy::unpack_results_buffer(), ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by Strategy::schedule_iterators().

13.147.3.12 **void serve_iterators (Iterator & the_iterator, Model & the_model)** [protected]

executed on the slave iterator servers to perform iterator jobs assigned by the strategy master (called by derived run_strategy()) This function is similar in structure to ApplicationInterface::serve_evaluations_synch().

References ParallelLibrary::bcast_i(), Strategy::iteratorCommRank, Strategy::iteratorCommSize, Strategy::pack_results_buffer(), ParallelLibrary::parallel_time(), Strategy::parallelLib, Strategy::paramsMsgLen, ParallelLibrary::irecv_si(), Strategy::resultsMsgLen, Strategy::run_iterator(), ParallelLibrary::send_si(), Strategy::unpack_parameters_buffer(), and Strategy::update_local_results().

Referenced by Strategy::schedule_iterators().
13.147 Strategy Class Reference

13.147.3.13 Strategy * get_strategy () [private]

Used by the envelope to instantiate the correct letter class. Used only by the envelope constructor to initialize strategyRep to the appropriate derived type, as given by the strategyName attribute.

References ProblemDescDB::get_string(), Strategy::probDescDB, Strategy::strategyName, and Dakota::strbegins().

Referenced by Strategy::Strategy().

The documentation for this class was generated from the following files:

- DakotaStrategy.hpp
- DakotaStrategy.cpp
13.148 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 String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  
  alternate constructor

- **SurfpackApproximation (const ProblemDescDB &problem_db, size_t num_vars)**

  standard constructor: Surfpack surface of appropriate type will be created

- **~SurfpackApproximation ()**

  destructor

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 ()**

  SurfData object will be created from Dakota's SurrogateData, and the appropriate Surfpack build method will be invoked.

- **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)

- 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 (const RealMatrix &challenge_points, int fn_index)
  compute and print all requested diagnostics for user provided challenge pts

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

- 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

• 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.

• String exportModelName
  A Surfpack model name for saving the surrogate model.

• 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

13.148.1 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 adaptative regression splines (MARS).
### 13.148.2 Constructor & Destructor Documentation

#### 13.148.2.1 SurfpackApproximation (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(), SurfpackApproximation::approxOrder, Approximation::approxType, Approximation::buildDataOrder, SurfpackApproximation::factory, and Approximation::outputLevel.

### 13.148.3 Member Function Documentation

#### 13.148.3.1 void build () [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(), Approximation::approxCLowerBnds, Approximation::approxCUpperBnds, Approximation::approxDILowerBnds, Approximation::approxDIUpperBnds, Approximation::approxDRLowerBnds, Approximation::approxDRUpperBnds, SurfpackApproximation::exportModelName, SurfpackApproximation::factory, SurfpackApproximation::merge_variable_arrays(), SurfpackApproximation::model, Approximation::outputLevel, SurfpackApproximation::surfData, and SurfpackApproximation::surrogates_to_surf_data().

#### 13.148.3.2 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, Approximation::approxType, Variables::cv(), SurfpackApproximation::model, and SurfpackApproximation::vars_to_realarray().

Referenced by SurfpackApproximation::add_anchor_to_surfdata(), and SurfpackApproximation::add_sd_to_surfdata().
13.148.3.3  **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(), SurfpackApproximation::add_sd_to_surfdata(), Approximation::approxData, Approximation::buildDataOrder, SurfpackApproximation::factory, and Approximation::outputLevel.

Referenced by SurfpackApproximation::build().

13.148.3.4  **void add_anchor_to_surfdata (SurfData & surf_data)  [private]**

Set the anchor point (including gradient and hessian if present) into surf_data 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(), SurfpackApproximation::copy_matrix(), SurfpackApproximation::gradient(), SurfpackApproximation::hessian(), Approximation::outputLevel, SurfpackApproximation::sdv_to_realarray(), and Dakota::write_-data().

Referenced by SurfpackApproximation::surrogates_to_surf_data().

The documentation for this class was generated from the following files:

- SurfpackApproximation.hpp
- SurfpackApproximation.cpp
13.149 SurrBasedGlobalMinimizer Class Reference

The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls. Inheritance diagram for SurrBasedGlobalMinimizer:

```
  SurrBasedGlobalMinimizer
  SurrBasedMinimizer
  Minimizer
  Iterator
```

### Public Member Functions

- **SurrBasedGlobalMinimizer (Model &model)**
  
  *constructor*

- **~SurrBasedGlobalMinimizer ()**
  
  *destructor*

### Protected Member Functions

- **bool returns_multiple_points () const**
  
  *Global surrogate-based methods can return multiple points.*

### Private Member Functions

- **void minimize_surrogates ()**
  
  *Performs global surrogate-based optimization by repeatedly optimizing on and improving surrogates of the response functions.*

### Private Attributes

- **bool replacePoints**
  
  *flag for replacing the previous iteration’s point additions, rather than continuing to append, during construction of the next surrogate*
13.149.1 Detailed Description

The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls. This method uses a SurrogateModel 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
13.150 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 (Model &model)**
  
  *constructor*

- **~SurrBasedLocalMinimizer ()**
  
  *destructor*

Protected Member Functions

- **void reset ()**
  
  *reset convergence controls in case of multiple SBLM executions*

Private Member Functions

- **void minimize_surrogates ()**
  
  *Performs local surrogate-based minimization by minimizing local, global, or hierarchical surrogates over a series of trust regions.*

- **bool tr_bounds (const RealVector &global_lower_bnds, const RealVector &global_upper_bnds, RealVector &tr_lower_bnds, RealVector &tr_upper_bnds)**
  
  *compute current trust region bounds*

- **void find_center_truth (const Iterator &dace_iterator, Model &truth_model)**
  
  *retrieve responseCenterTruth if possible, evaluate it if not*

- **void find_center_approx ()**
retrieve responseCenter_approx if possible, evaluate it if not

• void hard_convergence_check (const Response &response_truth, const RealVector &c_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)
  check for hard convergence (norm of projected gradient of merit function near zero)

• void tr_ratio_check (const RealVector &c_vars_star, const RealVector &tr_lower_bounds, const RealVector &tr_upper_bounds)
  compute trust region ratio (for SBLM iterate acceptance and trust region resizing) and check for soft convergence (diminishing returns)

• void update_penalty (const RealVector &fns_center_truth, const RealVector &fns_star_truth)
  initialize and update the penaltyParameter

• void relax_constraints (const RealVector &lower_bnds, const RealVector &upper_bnds)
  relax constraints by updating bounds when current iterate is infeasible

Static Private 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 RealVector &lower_bnds, const RealVector &upper_bnds, const RealVector &tr_lower_bounds, const RealVector &tr_upper_bounds)
  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 *needc, 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.

Private Attributes

• Real origTrustRegionFactor
  original user specification for trustRegionFactor

• Real trustRegionFactor
  the trust region factor is used to compute the total size of the trust region -- it is a percentage, e.g. for trustRegionFactor = 0.1, the actual size of the trust region will be 10% of the global bounds (upper bound - lower bound for each design variable).
• 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

• 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

• 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

• bool recastSubProb
  flag to indicate when approxSubProbModel involves a RecastModel recursion

• short trConstraintRelax
  type of trust region constraint relaxation for infeasible starting points: NO_RELAX or HOMOTOPY

• short meritFnType
  type of merit function used in trust region ratio logic: PENALTYMerit, ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, or AUGMENTED_LAGRANGIAN_MERIT

• short acceptLogic
  type of iterate acceptance test logic: FILTER or TR_RATIO

• int penaltyIterOffset
  iteration offset used to update the scaling of the penalty parameter for adaptive_penalty merit functions

• short convergenceFlag
  code indicating satisfaction of hard or soft convergence conditions
• short **softConvCount**
  
  number of consecutive candidate point rejections. If the count reaches **softConvLimit**, stop SBLM.

• short **softConvLimit**
  
  the limit on consecutive candidate point rejections. If exceeded by **softConvCount**, stop SBLM.

• bool **truthGradientFlag**
  
  flags the use/availability of truth gradients within the SBLM process

• bool **approxGradientFlag**
  
  flags the use/availability of surrogate gradients within the SBLM process

• bool **truthHessianFlag**
  
  flags the use/availability of truth Hessians within the SBLM process

• bool **approxHessianFlag**
  
  flags the use/availability of surrogate Hessians within the SBLM process

• short **correctionType**
  
  flags the use of surrogate correction techniques at the center of each trust region

• bool **globalApproxFlag**
  
  flags the use of a global data fit surrogate (rsm, ann, mars, kriging)

• bool **multiptApproxFlag**
  
  flags the use of a multipoint data fit surrogate (TANA)

• bool **localApproxFlag**
  
  flags the use of a local data fit surrogate (Taylor series)

• bool **hierarchApproxFlag**
  
  flags the use of a model hierarchy/multifidelity surrogate

• bool **newCenterFlag**
  
  flags the acceptance of a candidate point and the existence of a new trust region center

• bool **daceCenterPtFlag**
  
  flags the availability of the center point in the DACE evaluations for global approximations (CCD, Box-Behnken)

• bool **multiLayerBypassFlag**
  
  flags the simultaneous presence of two conditions: (1) additional layerings w/ actual_model (e.g., surrogateModel = layered/nested/layered -> actual_model = nested/layered), and (2) a user-specification to bypass all layerings within actual_model for the evaluation of truth data (responseCenterTruth and responseStarTruth).

• bool **useDerivsFlag**
  
  flag for the "use_derivatives" specification for which derivatives are to be evaluated at each DACE point in global surrogate builds.
• 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)

• Variables varsCenter
  variables at the trust region center

• Response responseCenterApprox
  approx response at trust region center

• Response responseStarApprox
  approx response at SBLM cycle minimum

• IntResponsePair responseCenterTruth
  truth response at trust region center

• IntResponsePair responseStarTruth
  truth response at SBLM cycle minimum

Static Private Attributes

• static SurrBasedLocalMinimizer * sblmInstance
  pointer to SBLM instance used in static member functions

13.150.1 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.
13.150.2 Member Function Documentation

13.150.2.1 void minimize_surrogates () [private, 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 varied according to the goodness of the agreement between the approximations and the true response functions.

Implements SurrBasedMinimizer.

References Dakota::abort_handler(), Iterator::active_set(), Response::active_set(), Model::active_variables(), Graphics::add_datapoint(), DiscrepancyCorrection::apply(), SurrBasedLocalMinimizer::approxGradientFlag, SurrBasedLocalMinimizer::approxHessianFlag, SurrBasedMinimizer::approxSubProbMinimizer, SurrBasedLocalMinimizer::approxSubProbModel, Iterator::bestResponseArray, Iterator::bestVariablesArray, Model::build_approximation(), Model::component_parallel_mode(), DiscrepancyCorrection::compute(), Model::compute_response(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Variables::continuous_variables(), SurrBasedLocalMinimizer::convergenceFlag, Variables::copy(), Dakota::copy_data(), SurrBasedLocalMinimizer::correctionType, Model::current_response(), Model::current_variables(), SurrBasedLocalMinimizer::daceCenterPtFlag, Dakota::dakota_graphics, Model::discrepancy_correction(), Model::evaluation_id(), SurrBasedLocalMinimizer::find_center_approx(), SurrBasedLocalMinimizer::find_center_truth(), SurrBasedLocalMinimizer::globalApproxFlag, SurrBasedLocalMinimizer::hard_convergence_check(), Iterator::is_null(), Iterator::iteratedModel, SurrBasedLocalMinimizer::localApproxFlag, SurrBasedLocalMinimizer::localApproxFlag, Iterator::maxIterations, SurrBasedLocalMinimizer::minTrustRegionFactor, SurrBasedLocalMinimizer::multiLayerBypassFlag, SurrBasedLocalMinimizer::multiApproxFlag, SurrBasedLocalMinimizer::newCenterFlag, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Iterator::numContinuousVars, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedLocalMinimizer::origNonlinIneqUpperBnds, SurrBasedLocalMinimizer::origNonlinEqTargets, SurrBasedLocalMinimizer::recastSubProb, SurrBasedLocalMinimizer::relax_constraints(), ActiveSet::request_values(), SurrBasedLocalMinimizer::reset(), Iterator::response_results(), SurrBasedLocalMinimizer::responseCenterApprox, SurrBasedLocalMinimizer::responseCenterTruth, SurrBasedLocalMinimizer::responseStarApprox, SurrBasedLocalMinimizer::responseStarTruth, Iterator::run_iterator(), Iterator::sampling_scheme(), SurrBasedLocalMinimizer::sbIterNum, SurrBasedLocalMinimizer::sblmInstance, SurrBasedLocalMinimizer::softConvCount, SurrBasedLocalMinimizer::softConvLimit, Model::subordinate_iterator(), Model::surrogate_model(), Model::surrogate_response_mode(), SurrBasedLocalMinimizer::tr_bound(), SurrBasedLocalMinimizer::tr_ratio_check(), SurrBasedLocalMinimizer::trConstraintRelax, SurrBasedLocalMinimizer::trustRegionFactor, Model::truth_model(), SurrBasedLocalMinimizer::truthGradientFlag, SurrBasedLocalMinimizer::truthHessianFlag, Response::update(), SurrBasedLocalMinimizer::useDerivsFlag, Iterator::variables_results(), and SurrBasedLocalMinimizer::varsCenter.

13.150.2.2 void hard_convergence_check (const Response & response_truth, const RealVector & c_vars, const RealVector & lower_bnds, const RealVector & upper_bnds) [private]

check for hard convergence (norm of projected gradient of merit function near zero) 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.
13.150 SurrogateBasedMinimizer Class Reference

References SurrogateBasedMinimizer::acceptLogic, SurrogateBasedMinimizer::approxSubProbObj, SurrogateBasedMinimizer::constraint_violation(), Minimizer::constraintTol, SurrogateBasedMinimizer::convergenceFlag, Iterator::convergenceTol, Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SurrogateBasedMinimizer::lagrangian_gradient(), SurrogateBasedMinimizer::meritFnType, Iterator::numContinuousVars, Minimizer::numNonlinearConstraints, SurrogateBasedMinimizer::origNonlinEqTargets, SurrogateBasedMinimizer::origNonlinIneqLowerBnds, SurrogateBasedMinimizer::origNonlinIneqUpperBnds, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), SurrogateBasedMinimizer::sbIterNum, SurrogateBasedLocalMinimizer::truthGradientFlag, SurrogateBasedMinimizer::update_augmented_lagrange_multipliers(), SurrogateBasedMinimizer::update_filter(), and SurrogateBasedMinimizer::update_lagrange_multipliers().

Referenced by SurrogateBasedLocalMinimizer::minimize_surrogates().

13.150.2.3 void tr_ratio_check (const RealVector & c_vars_star, const RealVector & tr_lower_bnds, const RealVector & tr_upper_bnds) [private]

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 SurrogateBasedLocalMinimizer::acceptLogic, SurrogateBasedLocalMinimizer::approxSubProbObj, SurrogateBasedMinimizer::augmented_lagrangian_merit(), SurrogateBasedMinimizer::constraint_violation(), Minimizer::constraintTol, Iterator::convergenceTol, SurrogateBasedMinimizer::etaSequence, Response::function_values(), SurrogateBasedMinimizer::gammaContract, SurrogateBasedMinimizer::gammaExpand, SurrogateBasedMinimizer::globalApproxFlag, Iterator::iteratedModel, SurrogateBasedMinimizer::lagrangian_merit(), SurrogateBasedMinimizer::meritFnType, SurrogateBasedMinimizer::newCenterFlag, Iterator::numContinuousVars, SurrogateBasedMinimizer::origNonlinEqTargets, SurrogateBasedMinimizer::origNonlinIneqLowerBnds, SurrogateBasedMinimizer::origNonlinIneqUpperBnds, SurrogateBasedMinimizer::penalty_merit(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), SurrogateBasedLocalMinimizer::responseCenterApprox, SurrogateBasedLocalMinimizer::responseCenterTruth, SurrogateBasedLocalMinimizer::responseStarApprox, SurrogateBasedLocalMinimizer::responseStarTruth, SurrogateBasedLocalMinimizer::softConvCount, SurrogateBasedLocalMinimizer::trRatioContractValue, SurrogateBasedLocalMinimizer::trRatioExpandValue, SurrogateBasedLocalMinimizer::trustRegionFactor, SurrogateBasedMinimizer::update_augmented_lagrange_multipliers(), SurrogateBasedMinimizer::update_filter(), and SurrogateBasedLocalMinimizer::update_penalty().

Referenced by SurrogateBasedLocalMinimizer::minimize_surrogates().

13.150.2.4 void update_penalty (const RealVector & fns_center_truth, const RealVector & fns_star_truth) [private]

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 SurrogateBasedMinimizer::alphaEta, SurrogateBasedLocalMinimizer::approxSubProbObj, SurrogateBasedMinimizer::constraint_violation(), Minimizer::constraintTol, SurrogateBasedMinimizer::eta, SurrogateBasedMinimizer::etaSequence, Iterator::iteratedModel, SurrogateBasedLocalMinimizer::meritFnType, Minimizer::objective(), SurrogateBasedLocalMinimizer::penaltyIterOffset, SurrogateBasedMinimizer::penaltyParameter,
Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and SurrBasedMinimizer::sblIterNum.

Referenced by SurrBasedLocalMinimizer::tr_ratio_check().

13.150.2.5 void approx_subprob_objective_eval (const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response) [static, private]

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_merit(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_value(), Response::function_values(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_gradient(). SurrBasedMinimizer::lagrangian_merit(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Minimizer::numUserPrimaryFns, Minimizer::objective(), Minimizer::objective_gradient(), SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinEqLowerBnds, SurrBasedMinimizer::origNonlinEqUpperBnds, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and SurrBasedLocalMinimizer::sblInstance.

Referenced by SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

13.150.2.6 void approx_subprob_constraint_eval (const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response) [static, private]

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_value(), Response::function_values(), Minimizer::numUserPrimaryFns, SurrBasedLocalMinimizer::responseCenterApprox, SurrBasedLocalMinimizer::sblInstance, and SurrBasedLocalMinimizer::varsCenter.

Referenced by SurrBasedLocalMinimizer::SurrBasedLocalMinimizer().

13.150.2.7 void hom_objective_eval (int & mode, int & n, double * tau_and_x, double & f, double * grad_f, int &) [static, private]

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().
13.150.2.8  void hom_constraint_eval (int & mode, int & ncnltn, int & n, int & nrowj, int * needc, double * tau_and_x, double * c, double * cjac, int & nstate)  [static, private]

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::compute_response(), Model::continuous_variables(), Model::current_response(), Response::function_gradients(), Response::function_values(), SurrBasedLocalMinimizer::nonlinEqTargetsSlack, SurrBasedLocalMinimizer::nonlinIneqLowerBndsSlack, 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
13.151 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 (Model &model)**  
  *constructor*

- **~SurrBasedMinimizer ()**  
  *destructor*

- **void initialize_graphics (bool graph_2d, bool tabular_data, const String &tabular_file)**  
  *initialize graphics customized for surrogate-based iteration*

- **void run ()**  
  *run portion of run_iterator; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

- **void print_results (std::ostream &s)**

- **virtual void minimize_surrogates ()=0**  
  *Used for computing the optimal solution using a surrogate-based approach. Redefines the Iterator::run() virtual function.*

- **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*

- **bool update_filter (const RealVector &fn_vals)**  
  *update a filter from a 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

• 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)
  compute an augmented Lagrangian function from a set of function values

• void augmented_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 &alag_grad)
  compute the gradient 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

• int sbIterNum
  surrogate-based minimization iteration number

• RealVectorList sbFilter
  Set of response function vectors defining a filter (objective vs. constraint violation) for iterate selection/rejection.

• RealVector lagrangeMult
  Lagrange multipliers for basic Lagrangian calculations.
• RealVector augLagrangeMult
  Lagrange multipliers for augmented Lagrangian calculations.

• Real penaltyParameter
  the penalization factor for violated constraints used in quadratic penalty calculations; increased in update_penalty()

• RealVector origNonlinIneqLowerBnds
  original nonlinear inequality constraint lower bounds (no relaxation)

• RealVector origNonlinIneqUpperBnds
  original nonlinear inequality constraint upper bounds (no relaxation)

• RealVector origNonlinEqTargets
  original nonlinear equality constraint targets (no relaxation)

• Real eta
  constant used in etaSequence updates

• Real alphaEta
  power for etaSequence updates when updating penalty

• Real betaEta
  power for etaSequence updates when updating multipliers

• Real etaSequence
  decreasing sequence of allowable constraint violation used in augmented Lagrangian updates (refer to Conn, Gould, and Toint, section 14.4)

13.151.1 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.

13.151.2 Member Function Documentation

13.151.2.1 void run () [inline, protected, virtual]

run portion of run_iterator; 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 SurrBasedMinimizer::minimize_surrogates().
13.151.2.2  void print_results (std::ostream & s)  [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,
Iterator::numFunctions, Minimizer::numUserPrimaryFns, Minimizer::optimizationFlag, ActiveSet::request_values(),
Dakota::strbegins(), Model::truth_model(), and Dakota::write_data_partial().

13.151.2.3  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::iteratedModel, SurrBasedMinimizer::lagrangeMult, Iterator::numContinuousVars,
Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns,
Minimizer::objective_gradient(), SurrBasedMinimizer::origNonlinEqLowerBnds,
SurrBasedMinimizer::origNonlinEqUpperBnds, Model::primary_response_fn_sense(), and Model::primary_-
response_fn_weights().
Referenced by SurrBasedLocalMinimizer::hard_convergence_check().

13.151.2.4  void update_augmented_lagrange_multipliers (const RealVector & fn_vals)  [protected]

initialize and update the Lagrange multipliers for augmented Lagrangian For the Rockafellar augmented La-
grangian, 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::bigRealBoundSize, SurrBasedMinimizer::etaSequence, Minimizer::numNonlinearEqConstraints,
Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, SurrBasedMinimizer::origNonlinEqLowerBnds,
SurrBasedMinimizer::origNonlinEqUpperBnds, Model::primary_response_fn_sense(), and Model::primary_-
response_fn_weights().
Referenced by SurrBasedLocalMinimizer::hard_convergence_check(), EffGlobalMinimizer::minimize_-
surrogates_on_model(), and SurrBasedLocalMinimizer::tr_ratio_check().

13.151.2.5  bool update_filter (const RealVector & fn_vals)  [protected]

update a filter from a set of function values Update the sbFilter with fn_vals if new iterate is non-dominated.
References SurrBasedMinimizer::constraint_violation(), Iterator::iteratedModel,
Minimizer::numNonlinearConstraints, Minimizer::objective(), Model::primary_response_fn_sense(),
Model::primary_response_fn_weights(), and SurrBasedMinimizer::sbFilter.
13.151.2.6 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 inequality/equality constraints. This implementation follows the convention in Vanderplaats with \( g \leq 0 \) 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::tr_ratio_check().

13.151.2.7 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 \( g \leq 0 \) and \( h = 0 \). The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

References SurrBasedMinimizer::augLagrangeMult, Minimizer::bigRealBoundSize, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, and SurrBasedMinimizer::penaltyParameter.

Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), EffGlobalMinimizer::get_best_sample(), EffGlobalMinimizer::minimize_surrogates_on_model(), and SurrBasedLocalMinimizer::tr_ratio_check().

13.151.2.8 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) \( p = f + r_p cv \).

References SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, Minimizer::objective(), and SurrBasedMinimizer::penaltyParameter.

Referenced by SurrBasedLocalMinimizer::tr_ratio_check().
13.151.2.9  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+^T g+ + h+^T 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::hard_convergence_check(), EffGlobalMinimizer::minimize_surrogates_on_model(), SurrBasedMinimizer::penalty_merit(), SurrBasedLocalMinimizer::relax_constraints(), SurrBasedLocalMinimizer::tr_ratio_check(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

The documentation for this class was generated from the following files:

- SurrBasedMinimizer.hpp
- SurrBasedMinimizer.cpp
13.152 SurrogateModel Class Reference

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel). Inheritance diagram for SurrogateModel:

```
  Model
    SurrogateModel
      DataFitSurrModel
      HierarchSurrModel
```

Protected Member Functions

- **SurrogateModel** (ProblemDescDB &problem_db)
  
  *constructor*

- **SurrogateModel** (ParallelLibrary &parallel_lib, const SharedVariablesData &svd, const ActiveSet &set, short output_level)
  
  *alternate constructor*

- **~SurrogateModel** ()
  
  *destructor*

- **Model & subordinate_model** ()
  
  *return truth_model()*

- short surrogate_response_mode () const
  
  *return responseMode*

- **DiscrepancyCorrection & discrepancy_correction** ()
  
  *return deltaCorr*

- void check_submodel_compatibility (const Model &sub_model)
  
  *verify compatibility between SurrogateModel attributes and attributes of the submodel (DataFitSurrModel::actualModel or HierarchSurrModel::highFidelityModel)*

- **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

Protected Attributes

- IntSet surrogateFnIndices
  for mixed response sets, this array specifies the response function subset that is approximated

- IntResponseMap surrResponseMap
  map of surrogate responses used in derived_synchronize() and derived_synchronize_nowait() functions

- 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.

- IntIntMap truthIdMap
  map from actualModel/highFidelityModel evaluation ids to DataFitSurrModel.hppierarchSurrModel ids

- IntIntMap surrIdMap
  map from approxInterface/lowFidelityModel evaluation ids to DataFitSurrModel.hppierarchSurrModel ids

- 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.

- short responseMode
  an enumeration that controls the response calculation mode in {DataFit,Hierarch}SurrModel approximate response computations

- size_t approxBuilds
  number of calls to build_approximation()

- 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.

• 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.

• 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.

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() 

13.152.1 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.
13.152.2 Member Function Documentation

13.152.2.1 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(), Model::continuous_lower_bounds(), Constraints::continuous_lower_bounds(), Model::continuous_upper_bounds(), Constraints::continuous_upper_bounds(), Variables::continuous_variables(), Constraints::copy(), Variables::copy(), Model::current_variables(), Model::currentVariables, Model::discrete_int_lower_bounds(), Model::discrete_int_upper_bounds(), Constraints::discrete_int_lower_bounds(), Variables::discrete_int_variables(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_lower_bounds(), Variables::discrete_real_variables(), Variables::inactive_continuous_variables(), Variables::inactive_discrete_int_variables(), Variables::inactive_discrete_real_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::referenceICVars, SurrogateModel::referenceDIVars, SurrogateModel::referenceDRVars, Dakota::strbegins(), Model::subordinate_model(), Model::surrogateType, Model::truth_model(), SurrogateModel::truthModelCons, SurrogateModel::truthModelVars, Model::user_defined_constraints(), Model::userDefinedConstraints, and Variables::view().

Referenced by HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), HierarchSurrModel::derived_compute_response(), and DataFitSurrModel::derived_compute_response().

13.152.3 Member Data Documentation

13.152.3.1 short responseMode [protected]

an enumeration that controls the response calculation mode in [DataFit,Hierarch]SurrModel approximate response computations SurrogateBasedLocalMinimizer toggles this mode since compute_correction() does not back out old corrections.

Referenced by HierarchSurrModel::component_parallel_mode(), HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), HierarchSurrModel::derived Compute_response(), DataFitSurrModel::derived_compute_response(), HierarchSurrModel::derived_set_communicators(), HierarchSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize_approx(), HierarchSurrModel::derived_synchronize_nowait(), DataFitSurrModel::derived_synchronize_nowait(), HierarchSurrModel::serve(), SurrogateModel::surrogate_response_mode(), HierarchSurrModel::surrogate_response_mode(), and DataFitSurrModel::surrogate_response_mode().
13.152.3.2 size_t approxBuilds [protected]

number of calls to build_approximation() used as a flag to automatically build the approximation if one of the derived compute_response functions is called prior to build_approximation().

Referenced by DataFitSurrModel::append_approximation(), DataFitSurrModel::approximation_coeffs(), HierarchSurrModel::build_approximation(), DataFitSurrModel::build_approximation(), HierarchSurrModel::derived_async_compute_response(), DataFitSurrModel::derived_async_compute_response(), HierarchSurrModel::derived_compute_response(), DataFitSurrModel::derived_compute_response(), DataFitSurrModel::pop_approximation(), DataFitSurrModel::update_actual_model(), DataFitSurrModel::update_approximation(), DataFitSurrModel::update_from_actual_model(), and HierarchSurrModel::update_model().

The documentation for this class was generated from the following files:

- SurrogateModel.hpp
- SurrogateModel.cpp
13.153 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_iterator_concurrency`)
- void `set_communicators_checks` (int `max_iterator_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 std::string &`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

13.153.1 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.

13.153.2 Member Function Documentation

13.153.2.1 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.
Reimplemented in GridApplicInterface.
References ApplicationInterface::completionSet, and SysCallApplicInterface::test_local_evaluations().
13.153.2.2  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.

Reimplemented in GridApplicInterface.

References Dakota::abort_handler(), Response::active_set(), ApplicationInterface::completionSet,
SysCallApplicInterface::failCountMap, ProcessApplicInterface::fileNameMap, Interface::final_eval_id_-tag(),
Dakota::lookup_by_eval_id(), ApplicationInterface::manage_failure(), ParamResponsePair::prp_parameters(),
ParamResponsePair::prp_response(), ProcessApplicInterface::read_results_files(), SysCallApplicInterface::sysCallSet,
and SysCallApplicInterface::system_call_file_test().

Referenced by SysCallApplicInterface::wait_local_evaluations().

13.153.2.3  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.

Reimplemented in GridApplicInterface.

References SysCallApplicInterface::spawn_analysis_to_shell().

13.153.2.4  void init_communicators_checks (int max_iterator_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().

13.153.2.5  void set_communicators_checks (int max_iterator_concurrency)  [inline, protected, virtual]

Process run-time issues as hard errors.

Reimplemented from ApplicationInterface.

References Dakota::abort_handler(), and ApplicationInterface::check_multiprocessor_analysis().

13.153.2.6  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::...
face::multipleParamsFiles, ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

Referenced by SysCallApplicInterface::create_evaluation_process().

13.153.2.7   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, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

Referenced by SysCallApplicInterface::create_evaluation_process().

13.153.2.8   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, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

Referenced by SysCallApplicInterface::create_evaluation_process(), SysCallApplicInterface::synchronous_local_analysis(), and GridApplicInterface::synchronous_local_analysis().

13.153.2.9   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, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::resultsFileName, CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.

Referenced by SysCallApplicInterface::create_evaluation_process().

The documentation for this class was generated from the following files:

- SysCallApplicInterface.hpp
- SysCallApplcInterface.cpp
13.154 TANA3Approximation Class Reference

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

Inheritance diagram for TANA3Approximation:

```
Approximation
```

    TANA3Approximation

Public Member Functions

- TANA3Approximation ()
  default constructor

- TANA3Approximation (ProblemDescDB &problem_db, size_t num_vars)
  standard constructor

- TANA3Approximation (size_t num_vars, short data_order, short output_level)
  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 ()
  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 ()
**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

**13.154.1 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.

**13.154.2 Member Function Documentation**

**13.154.2.1 void build()** [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::buildDataOrder, TANA3Approximation::find_scaled_coefficients(), TANA3Approximation::minX, Approximation::numVars, and TANA3Approximation::pExp.
13.154.2.2 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
13.155 TaylorApproximation Class Reference

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

```
Approximation

TaylorApproximation
```

Public Member Functions

- `TaylorApproximation ()`
  default constructor

- `TaylorApproximation (ProblemDescDB &problem_db, size_t num_vars)`
  standard constructor

- `TaylorApproximation (size_t num_vars, short data_order, short output_level)`
  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 ()`
  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
13.155.1 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 \( \text{grad}(x_c)'(x - x_c) \) for first- and second-order, and plus \( (x - x_c)' \text{Hess}(x_c)(x - x_c)/2 \) for second-order.

13.155.2 Member Function Documentation

13.155.2.1 void build () [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::buildDataOrder, and Approximation::numVars.

The documentation for this class was generated from the following files:

- TaylorApproximation.hpp
- TaylorApproximation.cpp
13.156 TestDriverInterface Class Reference

Inheritance diagram for TestDriverInterface::

```
+ TestDriverInterface
  + DirectApplicInterface
  + ApplicationInterface
  + Interface
```

**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*

- `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 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 mf_rosenbrock()
  alternate Rosenbrock formulations for < multifidelity or model form studies

- int gerstner()
  the isotropic/anisotropic Gerstner test function family

- int scalable_gerstner()
  scalable versions of the Gerstner test family

- int steel_column_cost()
  the steel_column_cost UQ/OUU test function

- 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*

- 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*
984 Class Documentation

- `int shubert ()`
  returns the N-D shubert function

- `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

- `int salinas ()`
  direct interface to the SALINAS structural dynamics code

- `int mc_api_run ()`
  direct interface to ModelCenter via API, HKIM 4/3/03

13.156.1 Detailed Description

Specialization of `DirectApplicInterface` to embed algebraic test function drivers directly in Dakota

13.156.2 Member Function Documentation

13.156.2.1 `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::cantilever(), TestDriverInterface::cyl_head(), DirectApplicInterface::driverTypeMap, TestDriverInterface::extended_rosenbrock(), TestDriverInterface::generalized_rosenbrock(), TestDriverInterface::gerstner(), TestDriverInterface::herbie(), TestDriverInterface::lf_rosenbrock(), TestDriverInterface::lf_short_column(), TestDriverInterface::log_ratio(), TestDriverInterface::mc_api_run(), TestDriverInterface::mf_rosenbrock(), TestDriverInterface::mf_short_column(), TestDriverInterface::mod_cantilever(), TestDriverInterface::multimodal(), TestDriverInterface::rosenbrock(), TestDriverInterface::salinas(), TestDriverInterface::scalable_gerstner(), TestDriverInterface::scalable_monomials(), TestDriverInterface::scalable_text_book(), TestDriverInterface::short_column(), TestDriverInterface::shubert(), TestDriverInterface::side_impact_cost(), TestDriverInterface::side_impact_perf(), TestDriverInterface::smooth_herbie(), TestDriverInterface::sobol_g_function(), TestDriverInterface::sobol_ishigami(), TestDriverInterface::sobol_rational(), TestDriverInterface::steel_column_cost(), TestDriverInterface::steel_column_perf(), TestDriverInterface::text_book(), TestDriverInterface::text_book1(), TestDriverInterface::text_book2(), TestDriverInterface::text_book3(), and TestDriverInterface::text_book_ouu().

13.156.2.2 `void herbie1D (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().
13.156.2.3  void smooth_herbie1D (size_t der_mode, Real xc_loc, std::vector<Real> & w_and_ders)  
[private]

1D components of smooth_herbie function 1D Smoothed Herbie = 1DHerbie minus the high frequency sine term, and its derivatives (apart from a multiplicative factor)
Referenced by TestDriverInterface::smooth_herbie().

13.156.2.4  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().

13.156.2.5  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().

13.156.2.6  int smooth_herbie ()  [private]

returns the N-D smooth herbie function N-D Smoothed Herbie function and its derivatives.
References  DirectApplicInterface::directFnASV,  DirectApplicInterface::directFnDVV,  DirectApplicInterface::numDerivVars,  DirectApplicInterface::numVars,  TestDriverInterface::separable_combine(),  TestDriverInterface::smooth_herbie1D(), and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

13.156.2.7  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_herbie().
13.156.2.8 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::analysisDriverIndex, Dakota::dc_ptr_int, DirectApplicInterface::directFnASV, Interface::fnLabels, DirectApplicInterface::fnVals, Dakota::mc_ptr_int, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADR, DirectApplicInterface::numFns, DirectApplicInterface::xC, DirectApplicInterface::xCLabels, DirectApplicInterface::xDI, DirectApplicInterface::xDILabels, DirectApplicInterface::xD, and DirectApplicInterface::xDRLabels.

Referenced by TestDriverInterface::derived_map_ac().

The documentation for this class was generated from the following files:

- TestDriverInterface.hpp
- TestDriverInterface.cpp
13.157 TrackerHTTP Class Reference

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

Public Member Functions

- TrackerHTTP ()
  
  default constructor is allowed, but doesn’t track methods used and outputs on all ranks

- TrackerHTTP (ProblemDescDB &problem_db, int world_rank=0)
  
  standard constructor with ProblemDescDB, rank

- ~TrackerHTTP ()
  
  destructor to free handles

- void post_start ()
  
  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::time_t &rawtime, const std::string &mode) 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) const
  
  POST separate location and query; datatopost= "name=daniel&project(curl)."

- 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::string `trackerLocation`
  base URL for the tracker

• std::string `proxyLocation`
  if empty, proxy may still be specified via environment variables (unlike default CURL behavior)

• 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
13.157 TrackerHTTP Class Reference

13.157.1 Detailed Description

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

13.157.2 Member Function Documentation

13.157.2.1 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.

13.157.2.2 void send_data_using_post (const std::string & datatopost) const [private]

POST separate location and query; datatopost="name=daniel&project=curl". separate location and query; datatopost="name=daniel&project=curl"
References TrackerHTTP::curlPtr, TrackerHTTP::outputLevel, and TrackerHTTP::trackerLocation.
Referenced by TrackerHTTP::post_finish(), and TrackerHTTP::post_start().
The documentation for this class was generated from the following files:

- TrackerHTTP.hpp
- TrackerHTTP.cpp
13.158 Variables Class Reference

Base class for the variables class hierarchy. Inheritance diagram for Variables::

```
Variables
    MixedVariables
    RelaxedVariables
```

Public Member Functions

- **Variables ()**
  
  *default constructor*

- **Variables (const ProblemDescDB &problem_db)**
  
  *standard constructor*

- **Variables (const SharedVariablesData &svd)**
  
  *alternate constructor for instantiations on the fly*

- **Variables (const Variables &vars)**
  
  *copy constructor*

- **virtual ~Variables ()**
  
  *destructor*

- **Variables operator= (const Variables &vars)**
  
  *assignment operator*

- **virtual void reshape (const SizetArray &vc_totals)**
  
  *reshapes an existing Variables object based on the incoming variablesComponents*

- **virtual void read (std::istream &s)**
  
  *read a variables object from an std::istream*

- **virtual void write (std::ostream &s) const**
  
  *write a variables object to an std::ostream*

- **virtual void write_aprepro (std::ostream &s) const**
  
  *write a variables object to an std::ostream in aprepro format*

- **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)
  read a variables object in tabular format from an istream

• virtual void write_tabular (std::ostream &s) const
  write a variables object in tabular format to an std::ostream

• virtual void read (BiStream &s)
  read a variables object from the binary restart stream

• virtual void write (BoStream &s) const
  write a variables object to the binary restart stream

• 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 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 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 adrv () const
  total number of discrete real vars

• const SharedVariablesData & shared_data () const
  return sharedVarsData

• SharedVariablesData & shared_data ()
  return 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 & \texttt{discrete\_int\_variables()} const
  \par
  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 \texttt{discrete\_int\_variables\_view()})

- void \texttt{discrete\_int\_variable(int di\_var, size\_t index)}
  \par
  set an active discrete integer variable

- void \texttt{discrete\_int\_variables(const IntVector &di\_vars)}
  \par
  set the active discrete integer variables

- Real \texttt{discrete\_real\_variable(size\_t index)} const
  \par
  return an active discrete real variable

- const RealVector & \texttt{discrete\_real\_variables()} const
  \par
  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 \texttt{discrete\_real\_variables\_view()})

- void \texttt{discrete\_real\_variable(Real dr\_var, size\_t index)}
  \par
  set an active discrete real variable

- void \texttt{discrete\_real\_variables(const RealVector &dr\_vars)}
  \par
  set the active discrete real variables

- void \texttt{active\_variables(const Variables &vars)}
  \par
  copy the active cv/div/drv variables from vars

- RealVector \texttt{continuous\_variables\_view()} const
  \par
  return a mutable view of the active continuous variables

- IntVector \texttt{discrete\_int\_variables\_view()} const
  \par
  return a mutable view of the active discrete integer variables

- RealVector \texttt{discrete\_real\_variables\_view()} const
  \par
  return a mutable view of the active discrete real variables

- StringMultiArrayConstView \texttt{continuous\_variable\_labels()} const
  \par
  return the active continuous variable labels

- void \texttt{continuous\_variable\_labels(StringMultiArrayConstView cv\_labels)}
  \par
  set the active continuous variable labels

- void \texttt{continuous\_variable\_label(const String &cv\_label, size\_t index)}
  \par
  set an active continuous variable label

- StringMultiArrayConstView \texttt{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_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

UShortMultiArrayConstView **discrete_int_variable_types** () const
return the active discrete integer variable types

UShortMultiArrayConstView **discrete_real_variable_types** () const
return the active discrete real variable types

SizetMultiArrayConstView **continuous_variable_ids** () const
return the active continuous variable position identifiers

const SizetArray & **relaxed_discrete_ids** () const
returns the set of discrete variable ids relaxed into a continuous array

const RealVector & **inactive_continuous_variables** () const
return the inactive continuous variables

void **inactive_continuous_variables** (const RealVector &ic_vars)
set the inactive continuous variables

const IntVector & **inactive_discrete_int_variables** () const
return the inactive discrete variables

void **inactive_discrete_int_variables** (const IntVector &idi_vars)
set the inactive discrete variables

const RealVector & **inactive_discrete_real_variables** () const
return the inactive discrete variables
• void **inactive_discrete_real_variables** (const RealVector &idr_vars)
  set the inactive discrete variables

• 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 integer variable labels

• void **inactive_discrete_int_variable_labels** (StringMultiArrayConstView idi_vars)
  set the inactive discrete integer variable labels

• StringMultiArrayConstView **inactive_discrete_real_variable_labels** () const
  return the inactive discrete real variable labels

• void **inactive_discrete_real_variable_labels** (StringMultiArrayConstView idr_vars)
  set the inactive discrete real 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_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

• 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

• 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_real_variable_labels () const
  returns a single array with all discrete variable labels

• void all_discrete_real_variable_labels (StringMultiArrayConstView adrv_labels)
  sets all discrete variable labels using a single array

• void all_discrete_real_variable_label (const String &adrv_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_real_variable_types () const
  return all discrete variable types

- SizetMultiArrayConstView all_continuous_variable_ids () const
  return all continuous variable position identifiers

Variables copy () const
  for use when a deep copy is needed (the representation is _not_ shared)

- 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)

- virtual void build_active_views ()
  construct active views of all variables arrays

- virtual void build_inactive_views ()
  construct inactive views of all variables arrays
void build_views()

construct active/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 (design, uncertain, state)

- IntVector allDiscreteIntVars
  array combining all of the discrete integer variables (design, state)

- RealVector allDiscreteRealVars
  array combining all of the discrete real variables (design, state)

- 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

**Private Attributes**

- `Variables * variablesRep`
  
  pointer to the letter (initialized only for the envelope)

- `int referenceCount`
  
  number of objects sharing `variablesRep`

**Friends**

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

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

- `std::size_t hash_value` (const `Variables &vars`)
  
  `hash_value`

- bool `binary_equal_to` (const `Variables &vars1`, const `Variables &vars2`)
  
  `binary_equal_to` (since `operator==` is not suitable for boost/hash_set)

### 13.158.1 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.

13.158.2 Constructor & Destructor Documentation

13.158.2.1 Variables ()

default constructor The default constructor: variablesRep is NULL in this case (a populated problem_db is needed to build a meaningful Variables object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

13.158.2.2 Variables (const ProblemDescDB & problem_db)

standard constructor This is the primary envelope constructor which uses problem_db to build a fully populated variables object. It only needs to extract enough data to properly execute get_variables(problem_db), since the constructor overloaded with BaseConstructor builds the actual base class data inherited by the derived classes. References Dakota::abort_handler(), Variables::get_variables(), and Variables::variablesRep.

13.158.2.3 Variables (const SharedVariablesData & svd)

alternate constructor for instantiations on the fly This is the alternate envelope constructor for instantiations on the fly. This constructor executes get_variables(view), which invokes the default derived/base constructors, followed by a resize() based on vars_comps.
References Dakota::abort_handler(), Variables::get_variables(), and Variables::variablesRep.

13.158.2.4 Variables (const Variables & vars)

copy constructor Copy constructor manages sharing of variablesRep and incrementing of referenceCount.
References Variables::referenceCount, and Variables::variablesRep.

13.158.2.5 ~Variables () [virtual]

destructor Destructor decrements referenceCount and only deletes variablesRep when referenceCount reaches zero.
References Variables::referenceCount, and Variables::variablesRep.

13.158.2.6 Variables (BaseConstructor, const ProblemDescDB & problem_db, const std::pair< short, short > & view) [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_variables() 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_variables()` again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in `~Variables`).

### 13.158.2.7 Variables (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_variables()` 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_variables()` again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in `~Variables`).

### 13.158.3 Member Function Documentation

#### 13.158.3.1 Variables operator= (const Variables & vars)


References Variables::referenceCount, and Variables::variablesRep.

#### 13.158.3.2 Variables copy () const

For use when a deep copy is needed (the representation is _not_ shared) Deep copies are used for history mechanisms such as bestVariablesArray and data_pairs since these must catalogue copies (and should not change as the representation within currentVariables changes).

References Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::build_views(), Variables::get_variables(), Variables::sharedVarsData, and Variables::variablesRep.

Referenced by Model::asynch_compute_response(), ApplicationInterface::continuation(), RecastModel::derived_asynch_compute_response(), HierarchSurrModel::derived_asynch_compute_response(), DataFitSurrModel::derived_asynch_compute_response(), EffGlobalMinimizer::EffGlobalMinimizer(), SurrogateModel::force_rebuild(), DiscrepancyCorrection::initialize_corrections(), LeastSq::LeastSq(), SurrBasedLocalMinimizer::minimize_surrogates(), Optimizer::Optimizer(), NonDLHSEvidence::post_process_samples(), COLINOptimizer::post_run(), Analyzer::read_variables_responses(), RecastModel::RecastModel(), Minimizer::resize_best_vars_array(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), Analyzer::update_best(), and NonDLocalReliability::update_mpp_search_data().

#### 13.158.3.3 void build_views () [inline, protected]

Construct active/inactive views of all variables arrays

```cpp
= EMPTY
= EMPTY
```
References Variables::build_active_views(), Variables::build_inactive_views(), Variables::sharedVarsData, SharedVariablesData::view(), and Variables::view().

Referenced by Variables::copy(), MixedVariables::MixedVariables(), Variables::read(), Variables::read_annotated(), RelaxedVariables::RelaxedVariables(), RelaxedVariables::reshape(), and MixedVariables::reshape().

13.158.3.4 Variables * get_variables (const ProblemDescDB & problem_db) [private]

Used by the standard envelope constructor to instantiate the correct letter class. Initializes variablesRep to the appropriate derived type, as given by problem_db attributes. The standard derived class constructors are invoked.

References Variables::get_view(), and Variables::view().

Referenced by Variables::copy(), Variables::read(), Variables::read_annotated(), and Variables::Variables().

13.158.3.5 Variables * get_variables (const SharedVariablesData & svd) const [private]

Used by the alternate envelope constructors, by read functions, and by copy() to instantiate a new letter class. Initializes variablesRep to the appropriate derived type, as given by view. The default derived class constructors are invoked.

References SharedVariablesData::view().

13.158.3.6 short method_map (short view_spec, bool relaxed) const [private]

infer domain from method selection Aggregate view and domain settings.

References Dakota::abort_handler().

Referenced by Variables::get_view().

The documentation for this class was generated from the following files:

- DakotaVariables.hpp
- DakotaVariables.cpp
13.159 Verification Class Reference

Base class for managing common aspects of verification studies. Inheritance diagram for Verification:

```
Verification
    |     /
    |    /  Iterator
    |   /   
    |  /    Analyzer
    | /     
    V      Verification
           |     /
           |    /   RichExtrapVerification
```

Protected Member Functions

- **Verification (Model &model)**
  
  *constructor*

- **Verification (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor for instantiations "on the fly"*

- **~Verification ()**
  
  *destructor*

- **void run ()**
  
  *run portion of run_iterator; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

- **void print_results (std::ostream &s)**
  
  *print the final iterator results*

- **virtual void perform_verification ()=0**
  
  *Redefines the run_iterator virtual function for the PStudy/DACE branch.*

13.159.1 Detailed Description

Base class for managing common aspects of verification studies. The Verification base class manages common data and functions, such as those involving ...
13.159.2 Member Function Documentation

13.159.2.1 void run () [inline, protected, virtual]

run portion of run_iterator; 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 Verification::perform_verification().

13.159.2.2 void print_results (std::ostream & s) [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.

The documentation for this class was generated from the following files:

- DakotaVerification.hpp
- DakotaVerification.cpp
14.1 dakota_dll_api.cpp File Reference

This file contains a DakotaRunner class, which launches DAKOTA.

Namespaces

- namespace Dakota
  
  *The primary namespace for DAKOTA.*

Functions

- void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, char *logname)
  
  *create and configure a new DakotaRunner, adding it to list of instances*

- int DAKOTA_DLL_FN dakota_readInput (int id, 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_mc_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

14.1.2.1 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, char *logname)
  
  *create and configure a new DakotaRunner, adding it to list of instances*

- int DAKOTA_DLL_FN dakota_readInput (int id, 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 current design point*

- 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*

- 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

14.2.2.1 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_tabular_io.hpp File Reference

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

- namespace Dakota
  
  The primary namespace for DAKOTA.

Functions

- 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 write_header_tabular (std::ostream &tabular_ostream, const Variables &vars, const Response &response, const std::string &counter_label, bool active_only=false, bool response_labels=true)
  
  output the header row (labels) for a tabular data file used by Analyzer and Graphics

- void write_data_tabular (std::ostream &tabular_ostream, const Variables &vars, const Response &response, size_t counter=_NPOS, bool active_only=false, bool write_responses=true)
  
  output a row of tabular data from variables and response object used by graphics to append to tabular file during iteration

- bool exists_extra_data (std::istream &tabular_file)
  
  Check if an input stream contains unexpected additional data.

- void read_header_tabular (std::istream &input_stream)
  
  read and discard header line from the stream

- void read_data_tabular (const std::string &input_filename, const std::string &context_message, RealVector &input_data, size_t num_entries, bool annotated)
  
  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 read_data_tabular (const std::string &input_filename, const std::string &context_message, RealArray &input_vector, bool annotated, size_t num_vars)
  
  read possibly header-annotated whitespace-separated data into a dynamic vector with minimal error checking
• void read_data_tabular (const std::string &input_filename, const std::string &context_message, VariablesList &input_vars, ResponseList &input Resp, const SharedVariablesData &svd, size_t num_c_vars, const ActiveSet &temp_set, bool annotated, bool verbose=false)
  
  read whitespace-separated data with optional row and column headers into lists of Variables (using provided SVD) and Responses until out of data; continuous variables only

• 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, bool annotated, bool verbose=false)
  
  read whitespace-separated data with optional row and column headers into a single matrix

14.3.1 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.
14.4 JEGAOptimizer.cpp File Reference

Contains the implementation of the JEGAOptimizer class.

Classes

- 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.*

- class **Driver**
  
  *A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.*

Namespaces

- namespace **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.*

14.4.1 Detailed Description

Contains the implementation of the JEGAOptimizer class.
14.5  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).

Namespaces

- namespace Dakota
  
  The primary namespace for DAKOTA.

14.5.1  Detailed Description

Contains the definition of the JEGAOptimizer class.
14.6 library_mode.cpp File Reference

code file containing a mock simulator main for testing DAKOTA in library mode

Namespaces

- namespace Dakota
  
  The primary namespace for DAKOTA.

Functions

- void nidr_set_input_string (const char*)
  
  Set input to NIDR via string argument instead of input file.

- int nidr_save_exedir (const char*, int)
- void run_dakota_parse (const char*dakota_input_file)
  
  Function to encapsulate the DAKOTA object instantiations for mode 1: parsing an input file.

- void run_dakota_data ()
  
  Function to encapsulate the DAKOTA object instantiations for mode 2: direct Data class instantiation.

- void run_dakota_mixed (const char*dakota_input_file)
  
  Function to encapsulate the DAKOTA object instantiations for mode 3: mixed parsing and direct updating.

- void model_interface_plugins (Dakota::ProblemDescDB &problem_db)
- int main (int argc, char*argv[])
  
  A mock simulator main for testing DAKOTA in library mode.

- static void my_callback_function (void*ptr)

14.6.1 Detailed Description

code file containing a mock simulator main for testing DAKOTA in library mode

14.6.2 Function Documentation

14.6.2.1 void run_dakota_parse (const char *dakota_input_file)

Function to encapsulate the DAKOTA object instantiations for mode 1: parsing an input file. This function parses from an input file to define the ProblemDescDB data.

References ProblemDescDB::lock(), ProblemDescDB::manage_inputs(), model_interface_plugins(), Strategy::run_strategy(), and ParallelLibrary::world_rank().

Referenced by main().

Dakota Version 5.4 Developers Manual generated on November 7, 2013
14.6.2.2  void run_dakota_mixed (const char ∗ dakota_input_file)

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 Strategy instantiation, (3) updates directly to Iterators/Models following Strategy instantiation.

References ProblemDescDB::broadcast(), ProblemDescDB::get_sa(), ProblemDescDB::lock(), model_interface_plugins(), ProblemDescDB::model_list(), ParallelLibrary::mpirun_flag(), my_callback_function(), nidr_set_input_string(), ProblemDescDB::parse_inputs(), ProblemDescDB::post_process(), ProblemDescDB::resolve_top_method(), Strategy::run_strategy(), ProblemDescDB::set(), and ParallelLibrary::world_rank().

Referenced by main().

14.6.2.3  void model_interface_plugins (Dakota::ProblemDescDB & problem_db)

Iterate over models and plugin appropriate interface: serial rosenbrock or parallel textbook.

References Dakota::abort_handler(), Interface::analysis_drivers(), Interface::assign_rep(), Dakota::contains(), Interface::interface_type(), ProblemDescDB::model_list(), ParallelLevel::server_intra_communicator(), and ProblemDescDB::set_db_model_nodes().

Referenced by Dakota::run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

14.6.2.4  int main (int argc, char ∗ argv[ ])

A mock simulator main for testing DAKOTA in library mode. 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 an input file (2) run_dakota_data: creates all problem specification from direct Data instance instantiations. (3) run_dakota_mixed: a mixture of input parsing (by file or default string) and direct data updates, where the data updates occur: (a) via the DB prior to Strategy instantiation, and (b) via Iterators/Models following Strategy instantiation. Usage: dakota_library_mode [-m] [dakota.in]

References ParallelLibrary::detect_parallel_launch(), Dakota::run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

14.6.2.5  static void my_callback_function (void ∗ ptr) [static]

Example of user-provided callback function to override input specified and managed by NIDR, e.g., from an input deck.

References Dakota::contains(), ProblemDescDB::get_sa(), ProblemDescDB::get_string(), ProblemDescDB::resolve_top_method(), and ProblemDescDB::set().

Referenced by run_dakota_mixed().
14.7 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.

- void nidr_set_input_string (const char *)
  
  Set input to NIDR via string argument instead of input file.

- 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.

14.7.1 Detailed Description

file containing a mock simulator main for testing DAKOTA in library mode on a split communicator
14.8 main.cpp File Reference

file containing the main program for DAKOTA

Functions

- void fpinit_ASL ()
- int nidr_save_exedir (const char *, int)
- int main (int argc, char *argv[])

The main DAKOTA program.

14.8.1 Detailed Description

file containing the main program for DAKOTA

14.8.2 Function Documentation

14.8.2.1 void fpinit_ASL ()

Floating-point initialization from AMPL: switch to 53-bit rounding if appropriate, to eliminate some cross-platform differences.

Referenced by main().

14.8.2.2 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 Strategy and invoke its run_strategy() virtual function.

References Dakota::abort_handler(), fpinit_ASL(), CommandLineHandler::instantiate_flag(), ProblemDescDB::lock(), ProblemDescDB::manage_inputs(), ParallelLibrary::output_helper(), GetLongOpt::retrieve(), Strategy::run_strategy(), ParallelLibrary::specify_outputs_restart(), and ParallelLibrary::world_rank().
14.9 restart_util.cpp File Reference

file containing the DAKOTA restart utility main program

Namespaces

- namespace Dakota

  *The primary namespace for DAKOTA.*

Functions

- void print restart (int argc, char **argv, String print_dest)
  * print a restart file

- void print_restart_tabular (int argc, char **argv, String print_dest)
  * print a restart file (tabular format)

- void read_neutral (int argc, char **argv)
  * read a restart file (neutral file format)

- void repair_restart (int argc, char **argv, String identifier_type)
  * repair a restart file by removing corrupted evaluations

- void concatenate_restart (int argc, char **argv)
  * concatenate multiple restart files

- int nidr_save_exedir (const char *, int)

- int main (int argc, char *argv[])
  * The main program for the DAKOTA restart utility.

14.9.1 Detailed Description

file containing the DAKOTA restart utility main program

14.9.2 Function Documentation

14.9.2.1 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_tabular(), Dakota::read_neutral(), and Dakota::repair_restart().
Index

~Approximation
   Dakota::Approximation, 269

~BiStream
   Dakota::BiStream, 290

~Constraints
   Dakota::Constraints, 330

~DataFitSurrModel
   Dakota::DataFitSurrModel, 340

~EffGlobalMinimizer
   Dakota::EffGlobalMinimizer, 428

~Interface
   Dakota::Interface, 489

~Iterator
   Dakota::Iterator, 500

~Model
   Dakota::Model, 565

~NonDAdaptiveSampling
   Dakota::NonDAdaptiveSampling, 637

~NonDGPImpSampling
   Dakota::NonDGPImpSampling, 672

~ProblemDescDB
   Dakota::ProblemDescDB, 815

~Strategy
   Dakota::Strategy, 933

~Variables
   Dakota::Variables, 994

_initPts
   Dakota::JEGAOptimizer, 514

_model
   Dakota::JEGAOptimizer::Evaluator, 444

add_data
   Dakota::ResultsDBAny, 870

add_datapoint
   Dakota::Graphics, 469

allContinuousIds
   Dakota::SharedVariablesDataRep, 902

anisotropic_order_to_dimension_preference
   Dakota::NonDIntegration, 683

append_approximation
   Dakota::ApproximationInterface, 277
   Dakota::DataFitSurrModel, 343

approx_subprob_constraint_eval
   Dakota::SurrBasedLocalMinimizer, 952

approx_subprob_objective_eval
   Dakota::SurrBasedLocalMinimizer, 952

approxBuilds
   Dakota::SurrogateModel, 963

Approximation
   Dakota::Approximation, 268, 269

APPSEvalMgr
   Dakota::APPSEvalMgr, 282

APPSOptimizer
   Dakota::APPSOptimizer, 284

array_insert
   Dakota::ResultsDBAny, 870

assess_reconstruction
   Dakota::EfficientSubspaceMethod, 432

assign_rep
   Dakota::Interface, 490
   Dakota::Iterator, 504
   Dakota::Model, 571

assign_streams
   Dakota::CommandLineHandler, 306

asstring
   Dakota, 133

asynchronous_local_analyses
   Dakota::ProcessHandleApplicInterface, 826

asynchronous_local_evaluations
   Dakota::ApplicationInterface, 258

asynchronous_local_evaluations_nowait

A
   Dakota::CONMINOptimizer, 320

abort_handler_t
   Dakota, 132

accepts_multiple_points
   Dakota::JEGAOptimizer, 513

actualModel
   Dakota::DataFitSurrModel, 347

add_anchor_to_surfdata
   Dakota::SurfpackApproximation, 942
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dakota::DOTOptimizer, 422</td>
<td>kw_10, 140</td>
</tr>
<tr>
<td>Constraints</td>
<td>kw_100, 164</td>
</tr>
<tr>
<td>Dakota::Constraints, 330</td>
<td>kw_101, 164</td>
</tr>
<tr>
<td>constraintValues</td>
<td>kw_102, 164</td>
</tr>
<tr>
<td>Dakota::CONMINOptimizer, 317</td>
<td>kw_103, 164</td>
</tr>
<tr>
<td>Dakota::DOTOptimizer, 422</td>
<td>kw_104, 165</td>
</tr>
<tr>
<td>converge_order</td>
<td>kw_105, 165</td>
</tr>
<tr>
<td>Dakota::RichExtrapVerification, 882</td>
<td>kw_106, 165</td>
</tr>
<tr>
<td>converge_qoi</td>
<td>kw_107, 165</td>
</tr>
<tr>
<td>Dakota::RichExtrapVerification, 882</td>
<td>kw_108, 165</td>
</tr>
<tr>
<td>copy</td>
<td>kw_109, 165</td>
</tr>
<tr>
<td>Dakota::Constraints, 331</td>
<td>kw_11, 140</td>
</tr>
<tr>
<td>Dakota::Variables, 995</td>
<td>kw_110, 166</td>
</tr>
<tr>
<td>copy_data</td>
<td>kw_111, 166</td>
</tr>
<tr>
<td>Dakota::DDACEDesignCompExp, 407</td>
<td>kw_112, 166</td>
</tr>
<tr>
<td>create_evaluation_process</td>
<td>kw_113, 166</td>
</tr>
<tr>
<td>Dakota::ProcessHandleApplicInterface, 825</td>
<td>kw_114, 167</td>
</tr>
<tr>
<td>create_plots_2d</td>
<td>kw_115, 167</td>
</tr>
<tr>
<td>Dakota::Graphics, 468</td>
<td>kw_116, 167</td>
</tr>
<tr>
<td>create_tabular_datastream</td>
<td>kw_117, 167</td>
</tr>
<tr>
<td>Dakota::Graphics, 469</td>
<td>kw_118, 168</td>
</tr>
<tr>
<td>CreateEvaluator</td>
<td>kw_119, 168</td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer::EvaluatorCreator, 445</td>
<td>kw_12, 140</td>
</tr>
<tr>
<td>CT</td>
<td>kw_120, 168</td>
</tr>
<tr>
<td>Dakota::CONMINOptimizer, 319</td>
<td>kw_121, 168</td>
</tr>
<tr>
<td>cv_index_map</td>
<td>kw_122, 168</td>
</tr>
<tr>
<td>Dakota::NestedModel, 591</td>
<td>kw_123, 169</td>
</tr>
<tr>
<td>DakFuncs0</td>
<td>kw_124, 169</td>
</tr>
<tr>
<td>Dakota, 137</td>
<td>kw_125, 169</td>
</tr>
<tr>
<td>Dakota, 59</td>
<td>kw_126, 169</td>
</tr>
<tr>
<td>abort_handler_t, 132</td>
<td>kw_127, 169</td>
</tr>
<tr>
<td>asstring, 133</td>
<td>kw_128, 170</td>
</tr>
<tr>
<td>CAUVLbl, 228</td>
<td>kw_129, 170</td>
</tr>
<tr>
<td>CEUVLbl, 229</td>
<td>kw_13, 141</td>
</tr>
<tr>
<td>concatenate_restart, 136</td>
<td>kw_130, 170</td>
</tr>
<tr>
<td>DakFuncs0, 137</td>
<td>kw_131, 170</td>
</tr>
<tr>
<td>DAUVLbl, 229</td>
<td>kw_132, 171</td>
</tr>
<tr>
<td>DAURVLbl, 229</td>
<td>kw_133, 171</td>
</tr>
<tr>
<td>DEUVLbl, 229</td>
<td>kw_134, 171</td>
</tr>
<tr>
<td>DEURVLbl, 230</td>
<td>kw_135, 171</td>
</tr>
<tr>
<td>DiscSetLbl, 230</td>
<td>kw_136, 171</td>
</tr>
<tr>
<td>FIELD_NAMES, 137</td>
<td>kw_137, 172</td>
</tr>
<tr>
<td>flush, 131</td>
<td>kw_138, 172</td>
</tr>
<tr>
<td>get_cwd, 132</td>
<td>kw_139, 173</td>
</tr>
<tr>
<td>get_npath, 132</td>
<td>kw_14, 141</td>
</tr>
<tr>
<td>getdist, 132</td>
<td>kw_140, 173</td>
</tr>
<tr>
<td>getRmax, 133</td>
<td>kw_141, 173</td>
</tr>
<tr>
<td>id_vars_exact_compare, 134</td>
<td>kw_142, 173</td>
</tr>
<tr>
<td>kw_1, 137</td>
<td>kw_143, 173</td>
</tr>
<tr>
<td>kw_144, 174</td>
<td>kw_189, 185</td>
</tr>
<tr>
<td>kw_145, 174</td>
<td>kw_19, 142</td>
</tr>
<tr>
<td>kw_146, 174</td>
<td>kw_190, 185</td>
</tr>
<tr>
<td>kw_147, 174</td>
<td>kw_191, 186</td>
</tr>
<tr>
<td>kw_148, 174</td>
<td>kw_192, 186</td>
</tr>
<tr>
<td>kw_149, 175</td>
<td>kw_193, 186</td>
</tr>
<tr>
<td>kw_15, 141</td>
<td>kw_194, 186</td>
</tr>
<tr>
<td>kw_150, 175</td>
<td>kw_195, 187</td>
</tr>
<tr>
<td>kw_151, 176</td>
<td>kw_196, 187</td>
</tr>
<tr>
<td>kw_152, 176</td>
<td>kw_197, 188</td>
</tr>
<tr>
<td>kw_153, 176</td>
<td>kw_198, 188</td>
</tr>
<tr>
<td>kw_154, 176</td>
<td>kw_199, 189</td>
</tr>
<tr>
<td>kw_155, 177</td>
<td>kw_2, 138</td>
</tr>
<tr>
<td>kw_156, 177</td>
<td>kw_20, 142</td>
</tr>
<tr>
<td>kw_157, 177</td>
<td>kw_200, 189</td>
</tr>
<tr>
<td>kw_158, 177</td>
<td>kw_201, 189</td>
</tr>
<tr>
<td>kw_159, 178</td>
<td>kw_202, 190</td>
</tr>
<tr>
<td>kw_16, 141</td>
<td>kw_203, 190</td>
</tr>
<tr>
<td>kw_160, 178</td>
<td>kw_204, 190</td>
</tr>
<tr>
<td>kw_161, 178</td>
<td>kw_205, 190</td>
</tr>
<tr>
<td>kw_162, 178</td>
<td>kw_206, 191</td>
</tr>
<tr>
<td>kw_163, 178</td>
<td>kw_207, 191</td>
</tr>
<tr>
<td>kw_164, 178</td>
<td>kw_208, 191</td>
</tr>
<tr>
<td>kw_165, 179</td>
<td>kw_209, 191</td>
</tr>
<tr>
<td>kw_166, 179</td>
<td>kw_21, 142</td>
</tr>
<tr>
<td>kw_167, 179</td>
<td>kw_210, 191</td>
</tr>
<tr>
<td>kw_168, 180</td>
<td>kw_211, 192</td>
</tr>
<tr>
<td>kw_169, 180</td>
<td>kw_212, 193</td>
</tr>
<tr>
<td>kw_17, 142</td>
<td>kw_213, 193</td>
</tr>
<tr>
<td>kw_170, 180</td>
<td>kw_214, 193</td>
</tr>
<tr>
<td>kw_171, 180</td>
<td>kw_215, 194</td>
</tr>
<tr>
<td>kw_172, 181</td>
<td>kw_216, 194</td>
</tr>
<tr>
<td>kw_173, 181</td>
<td>kw_217, 194</td>
</tr>
<tr>
<td>kw_174, 181</td>
<td>kw_218, 194</td>
</tr>
<tr>
<td>kw_175, 181</td>
<td>kw_219, 195</td>
</tr>
<tr>
<td>kw_176, 182</td>
<td>kw_22, 143</td>
</tr>
<tr>
<td>kw_177, 182</td>
<td>kw_221, 195</td>
</tr>
<tr>
<td>kw_178, 182</td>
<td>kw_222, 196</td>
</tr>
<tr>
<td>kw_179, 182</td>
<td>kw_223, 196</td>
</tr>
<tr>
<td>kw_18, 142</td>
<td>kw_224, 196</td>
</tr>
<tr>
<td>kw_180, 182</td>
<td>kw_225, 196</td>
</tr>
<tr>
<td>kw_181, 183</td>
<td>kw_226, 196</td>
</tr>
<tr>
<td>kw_182, 184</td>
<td>kw_227, 197</td>
</tr>
<tr>
<td>kw_183, 184</td>
<td>kw_228, 197</td>
</tr>
<tr>
<td>kw_184, 184</td>
<td>kw_229, 197</td>
</tr>
<tr>
<td>kw_185, 184</td>
<td>kw_23, 143</td>
</tr>
<tr>
<td>kw_186, 185</td>
<td>kw_230, 197</td>
</tr>
<tr>
<td>kw_187, 185</td>
<td>kw_231, 198</td>
</tr>
<tr>
<td>kw_188, 185</td>
<td>kw_232, 198</td>
</tr>
<tr>
<td>kw_233</td>
<td>198</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
</tr>
<tr>
<td>kw_234</td>
<td>198</td>
</tr>
<tr>
<td>kw_235</td>
<td>199</td>
</tr>
<tr>
<td>kw_236</td>
<td>199</td>
</tr>
<tr>
<td>kw_237</td>
<td>199</td>
</tr>
<tr>
<td>kw_238</td>
<td>199</td>
</tr>
<tr>
<td>kw_239</td>
<td>199</td>
</tr>
<tr>
<td>kw_24</td>
<td>143</td>
</tr>
<tr>
<td>kw_240</td>
<td>200</td>
</tr>
<tr>
<td>kw_241</td>
<td>200</td>
</tr>
<tr>
<td>kw_242</td>
<td>200</td>
</tr>
<tr>
<td>kw_243</td>
<td>200</td>
</tr>
<tr>
<td>kw_244</td>
<td>201</td>
</tr>
<tr>
<td>kw_245</td>
<td>201</td>
</tr>
<tr>
<td>kw_246</td>
<td>202</td>
</tr>
<tr>
<td>kw_247</td>
<td>202</td>
</tr>
<tr>
<td>kw_248</td>
<td>202</td>
</tr>
<tr>
<td>kw_249</td>
<td>202</td>
</tr>
<tr>
<td>kw_25</td>
<td>143</td>
</tr>
<tr>
<td>kw_250</td>
<td>203</td>
</tr>
<tr>
<td>kw_251</td>
<td>203</td>
</tr>
<tr>
<td>kw_252</td>
<td>203</td>
</tr>
<tr>
<td>kw_253</td>
<td>204</td>
</tr>
<tr>
<td>kw_254</td>
<td>204</td>
</tr>
<tr>
<td>kw_255</td>
<td>205</td>
</tr>
<tr>
<td>kw_256</td>
<td>205</td>
</tr>
<tr>
<td>kw_257</td>
<td>205</td>
</tr>
<tr>
<td>kw_258</td>
<td>205</td>
</tr>
<tr>
<td>kw_259</td>
<td>206</td>
</tr>
<tr>
<td>kw_26</td>
<td>143</td>
</tr>
<tr>
<td>kw_260</td>
<td>206</td>
</tr>
<tr>
<td>kw_261</td>
<td>206</td>
</tr>
<tr>
<td>kw_262</td>
<td>207</td>
</tr>
<tr>
<td>kw_263</td>
<td>207</td>
</tr>
<tr>
<td>kw_264</td>
<td>208</td>
</tr>
<tr>
<td>kw_265</td>
<td>208</td>
</tr>
<tr>
<td>kw_266</td>
<td>208</td>
</tr>
<tr>
<td>kw_267</td>
<td>208</td>
</tr>
<tr>
<td>kw_268</td>
<td>209</td>
</tr>
<tr>
<td>kw_269</td>
<td>209</td>
</tr>
<tr>
<td>kw_27</td>
<td>144</td>
</tr>
<tr>
<td>kw_270</td>
<td>209</td>
</tr>
<tr>
<td>kw_271</td>
<td>210</td>
</tr>
<tr>
<td>kw_272</td>
<td>210</td>
</tr>
<tr>
<td>kw_273</td>
<td>210</td>
</tr>
<tr>
<td>kw_274</td>
<td>210</td>
</tr>
<tr>
<td>kw_275</td>
<td>211</td>
</tr>
<tr>
<td>kw_276</td>
<td>211</td>
</tr>
<tr>
<td>kw_277</td>
<td>211</td>
</tr>
</tbody>
</table>
INDEX

Dakota Version 5.4 Developers Manual generated on November 7, 2013

start_dakota_heartbeat, 131
start_grid_computing, 133
stop_grid_computing, 133
var_mp_cbound, 233
var_mp_check_cau, 231
var_mp_check_ceu, 232
var_mp_check_cv, 231
var_mp_check_daui, 232
var_mp_check_daur, 232
var_mp_check_deui, 232
var_mp_check_deur, 232
var_mp_check_dset, 231
var_mp_drange, 233
Vlch, 230
VLI, 231
VLR, 230
Dakota::ActiveSet, 235
derivVarsVector, 237
requestVector, 237
Dakota::Analyzer, 239
evaluate_parameter_sets, 243
pre_output, 242
print_results, 242
print_sobol_indices, 244
read_variables_responses, 243
variance_based_decomp, 243
Dakota::ApplicationInterface, 245
asynchronous_local_evaluations, 258
asynchronous_local_evaluations_nowait, 260
duplication_detect, 257
init_communicators_checks, 256
init_serial, 254
map, 254
master_dynamic_schedule_analyses, 256
master_dynamic_schedule_evaluations, 257
master_dynamic_schedule_evaluations_nowait, 259
peer_dynamic_schedule_evaluations, 258
peer_dynamic_schedule_evaluations_nowait, 259
peer_static_schedule_evaluations, 257
serve_analyses_synch, 256
serve_evaluations, 255
serve_evaluations_asynch, 261
serve_evaluations_asynch_peer, 261
serve_evaluations_synch, 260
serve_evaluations_synch_peer, 260
set_communicators_checks, 256
stop_evaluation_servers, 255
synch, 254
Dakota::Approximation, 263
~Approximation, 269
Approximation, 268, 269
build, 269
buildDataOrder, 271
clear_all, 271
clear_current, 271
finalize, 270
get_approx, 271
operator=<, 269
pop, 270
rebuild, 270
restore, 270
Dakota::ApproximationInterface, 273
append_approximation, 277
build_approximation, 277
functionSurfaces, 278
pop_approximation, 278
rebuild_approximation, 278
restore_approximation, 278
restore_available, 278
update_approximation, 276
Dakota::APPSEvalMgr, 280
APPSEvalMgr, 282
isReadyForWork, 282
recv, 282
submit, 282
Dakota::APPSOptimizer, 283
APPSOptimizer, 284
find_optimum, 285
initialize_variables_and_constraints, 285
set_apps_parameters, 285
Dakota::BaseConstructor, 287
Dakota::BiStream, 288
~BiStream, 290
BiStream, 289
operator>>, 290
Dakota::BoStream, 291
BoStream, 292
operator<<, 293
Dakota::COLINApplication, 294
colin_request_to_dakota_request, 296
collect_evaluation_impl, 296
dakota_response_to_colin_response, 297
evaluation_available, 296
map_domain, 297
perform_evaluation_impl, 296
INDEX

set_problem, 295
spawn_evaluation_impl, 295
Dakota::COLINOptimizer, 298
colin_cache_lookup, 302
COLINOptimizer, 300
constraintViolation, 302
find_optimum, 300
post_run, 301
returns_multiple_points, 301
set_rng, 301
set_solver_parameters, 301
solver_setup, 301
Dakota::CollaborativeHybridStrategy, 303
Dakota::CommandLineHandler, 305
assign_streams, 306
instantiate_flag, 306
output_helper, 307
output_version, 306
Dakota::CommandShell, 308
flush, 309
operator<<, 309
Dakota::ConcurrentStrategy, 310
pack_parameters_buffer, 311
pack_results_buffer, 312
unpack_parameters_buffer, 312
unpack_results_buffer, 312
Dakota::CONMINOptimizer, 313
A, 320
B, 319
C, 319
conminInfo, 317
constraintMappingIndices, 317
constraintMappingMultipliers, 318
constraintMappingOffsets, 318
constraintValues, 317
CT, 319
DF, 320
G1, 319
G2, 319
IC, 320
ISC, 320
MS1, 319
N1, 318
N2, 318
N3, 318
N4, 318
N5, 318
optimizationType, 317
printControl, 317
S, 319
SCAL, 320
Dakota::Constraints, 321
~Constraints, 330
build_views, 332
Constraints, 330
copy, 331
get_constraints, 332
manage_linear_constraints, 332
operator=, 331
reshape, 331
Dakota::DataFitSurrModel, 334
~DataFitSurrModel, 340
actualModel, 347
append_approximation, 343
build_approximation, 341
build_global, 344
build_local_multipoint, 345
derived_asynch_compute_response, 340
derived_compute_response, 340
derived_init_communicators, 344
derived_synchronize, 340
derived_synchronize_nowait, 341
evaluation_id, 344
import_points, 344
update_actual_model, 345
update_approximation, 342
update_from_actual_model, 346
Dakota::DataInterface, 348
Dakota::DataMethod, 350
Dakota::DataMethodRep, 352
Dakota::DataModel, 368
Dakota::DataModelRep, 370
Dakota::DataResponses, 376
Dakota::DataResponsesRep, 378
Dakota::DataStrategy, 383
Dakota::DataStrategyRep, 385
Dakota::DataVariables, 388
Dakota::DataVariablesRep, 390
Dakota::DDACEDesignCompExp, 403
copy_data, 407
DDACEDesignCompExp, 405
num_samples, 406
post_run, 406
pre_run, 406
resolve_samples_symbols, 406
Dakota::DirectApplicInterface, 408
derived_map_ac, 412
init_communicators_checks, 412
set_communicators_checks, 412
synchronous_local_analysis, 412
Dakota::DiscrepancyCorrection, 414
compute, 417
Dakota::DOTOptimizer, 419
constraintMappingIndices, 422
constraintMappingMultipliers, 422
constraintMappingOffsets, 422
constraintValues, 422
dotFDInfo, 421
dotInfo, 421
dotMethod, 421
intCntlParmArray, 422
printControl, 422
realCntlParmArray, 422
Dakota::EffGlobalMinimizer, 426
~EffGlobalMinimizer, 428
get_best_sample, 428
Dakota::EfficientSubspaceMethod, 429
assess_reconstruction, 432
map_xi_to_x, 432
reduced_space_uq, 432
uncertain_vars_to_subspace, 432
Dakota::EmbeddedHybridStrategy, 434
Dakota::ExperimentData, 447
load_scalar, 447
Dakota::ForkApplicInterface, 448
Dakota::FSUDesignCompExp, 451
enforce_input_rules, 454
FSUDesignCompExp, 453
num_samples, 454
post_run, 454
pre_run, 454
Dakota::GaussProcApproximation, 456
GPmodel_apply, 461
trendOrder, 461
Dakota::GetLongOpt, 463
enroll, 465
GetLongOpt, 465
MandatoryValue, 465
OptionalValue, 464
OptType, 464
parse, 465
retrieve, 465
usage, 466
Valueless, 464
Dakota::Graphics, 467
add_datapoint, 469
create_plots_2d, 468
create_tabular_datastream, 469
new_dataset, 469
Dakota::GridApplicInterface, 471
synchronous_local_analysis, 472
wait_local_evaluations, 472
Dakota::HierarchSurrModel, 474
derived_asynch_compute_response, 477
derived_compute_response, 477
derived_synchronize, 477
derived_synchronize_nowait, 478
evaluation_id, 478
Dakota::HybridStrategy, 479
Dakota::Interface, 481
~Interface, 489
assign_rep, 490
eval_tag_prefix, 490
get_interface, 491
Interface, 489
operator=, 490
rawResponseMap, 491
response_mapping, 490
Dakota::Iterator, 492
~Iterator, 500
assign_rep, 504
eval_tag_prefix, 504
fdGradStepSize, 505
fdHessByFnStepSize, 505
fdHessByGradStepSize, 505
finalize_run, 502
get_iterator, 505
initialize_graphics, 502
initialize_run, 501
Iterator, 499–501
num_samples, 503
operator=, 501
post_run, 502
pre_run, 501
print_results, 503
run, 502
run_iterator, 503
Dakota::JEGAOptimizer, 507
_initPts, 514
accepts_multiple_points, 513
find_optimum, 513
GetBestMOSolutions, 512
GetBestSolutions, 512
GetBestSOSSolutions, 512
initial_points, 513, 514
INDEX

JEGAOptimizer, 509
LoadAlgorithmConfig, 510
LoadDakotaResponses, 510
LoadProblemConfig, 510
LoadTheConstraints, 511
LoadTheDesignVariables, 511
LoadTheObjectiveFunctions, 511
LoadTheParameterDatabase, 510
returns_multiple_points, 513
ToDoubleMatrix, 512
Dakota::JEGAOptimizer::Driver, 424
DestroyAlgorithm, 425
Driver, 424
ExtractAllData, 424
PerformIterations, 425
Dakota::JEGAOptimizer::Evaluator, 438
_model, 444
Clone, 443
Description, 440
Evaluate, 442
Evaluator, 439, 440
GetDescription, 443
GetName, 443
GetNumberOfLinearConstraints, 442
GetNumberOfNonLinearConstraints, 441
Name, 440
RecordResponses, 441
SeparateVariables, 441
Dakota::JEGAOptimizer::EvaluatorCreator, 445
CreateEvaluator, 445
EvaluatorCreator, 445
Dakota::LeastSq, 515
finalize_run, 518
get_confidence_intervals, 518
initialize_run, 517
LeastSq, 517
post_run, 517
primary_resp_weighter, 519
print_results, 518
run, 517
weight_model, 518
Dakota::MatlabInterface, 520
derived_map_ac, 521
matlab_engine_run, 521
Dakota::Minimizer, 522
data_transform_model, 530
finalize_run, 530
gnewton_set_recast, 531
initialize_run, 529
initialize_scaling, 534
lin_coeffs_modify_n2s, 535
Minimizer, 529
modify_n2s, 535
modify_s2n, 531
need_responds, 531
objective, 532
objective_gradient, 532
objective_hessian, 533
primary_response_differencer, 533
resize_best_response_array, 533
resize_best_vars_array, 533
response_modify_n2s, 535
responseModify_s2n, 531
scale_model, 530
secondary_response_copier, 531
secondary_response_scaler, 534
variables_scaler, 534
Dakota::MixedVarConstraints, 536
MixedVarConstraints, 537
reshape, 537
Dakota::MixedVariables, 538
MixedVariables, 539
read_tabular, 539
Dakota::Model, 540
~Model, 565
assign_rep, 571
derivative_concurrency, 571
estimate_derivatives, 572
estimate_message_lengths, 570
eval_tag_prefix, 569
evaluation_cache, 569
FDstep1, 572
FDstep2, 572
get_model, 572
init_communicators, 570
init_serial, 570
initialize_h, 572
interface, 568
interface_id, 569
local_eval_concurrency, 568
local_eval_synchronization, 568
manage_asv, 574
Model, 565, 566
operator=, 566
subordinate_iterator, 566
subordinate_model, 566
subordinate_models, 569
surrogate_model, 567
<table>
<thead>
<tr>
<th>Index Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronize_derivatives</td>
<td>573</td>
</tr>
<tr>
<td>truth_model</td>
<td>567</td>
</tr>
<tr>
<td>update_from_subordinate_model</td>
<td>567</td>
</tr>
<tr>
<td>update_quasi_hessians</td>
<td>574</td>
</tr>
<tr>
<td>update_response</td>
<td>573</td>
</tr>
<tr>
<td>Dakota::MPIPackBuffer</td>
<td>575</td>
</tr>
<tr>
<td>Dakota::MPIUnpackBuffer</td>
<td>578</td>
</tr>
<tr>
<td>Dakota::NCSUOptimizer</td>
<td>581</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>583</td>
</tr>
<tr>
<td>objective_eval</td>
<td>584</td>
</tr>
<tr>
<td>Dakota::NestedModel</td>
<td>585</td>
</tr>
<tr>
<td>ccv_index_map</td>
<td>592</td>
</tr>
<tr>
<td>cdv_index_map</td>
<td>592</td>
</tr>
<tr>
<td>cdrv_index_map</td>
<td>592</td>
</tr>
<tr>
<td>cv_index_map</td>
<td>591</td>
</tr>
<tr>
<td>derived_asynch_compute_response</td>
<td>590</td>
</tr>
<tr>
<td>derived_compute_response</td>
<td>590</td>
</tr>
<tr>
<td>derived_init_communicators</td>
<td>591</td>
</tr>
<tr>
<td>derived_master_overload</td>
<td>591</td>
</tr>
<tr>
<td>div_index_map</td>
<td>592</td>
</tr>
<tr>
<td>drv_index_map</td>
<td>592</td>
</tr>
<tr>
<td>evaluation_id</td>
<td>591</td>
</tr>
<tr>
<td>response_mapping</td>
<td>593</td>
</tr>
<tr>
<td>subModel</td>
<td>594</td>
</tr>
<tr>
<td>Dakota::NIDRProblemDescDB</td>
<td>595</td>
</tr>
<tr>
<td>derived_parse_inputs</td>
<td>598</td>
</tr>
<tr>
<td>Dakota::NL2Res</td>
<td>600</td>
</tr>
<tr>
<td>Dakota::NL2SOLLeastSq</td>
<td>601</td>
</tr>
<tr>
<td>minimize_residuals</td>
<td>603</td>
</tr>
<tr>
<td>Dakota::NLQLPOptimizer</td>
<td>604</td>
</tr>
<tr>
<td>Dakota::NLSSOLLeastSq</td>
<td>610</td>
</tr>
<tr>
<td>NLSOLLeastSq</td>
<td>611</td>
</tr>
<tr>
<td>Dakota::NoDBaseConstructor</td>
<td>612</td>
</tr>
<tr>
<td>Dakota::NomadOptimizer</td>
<td>613</td>
</tr>
<tr>
<td>load_parameters</td>
<td>615</td>
</tr>
<tr>
<td>NomadOptimizer</td>
<td>615</td>
</tr>
<tr>
<td>Dakota::NomadOptimizer::Evaluator</td>
<td>436</td>
</tr>
<tr>
<td>eval_x</td>
<td>437</td>
</tr>
<tr>
<td>Evaluator</td>
<td>437</td>
</tr>
<tr>
<td>Dakota::NonD</td>
<td>616</td>
</tr>
<tr>
<td>finalize_run</td>
<td>625</td>
</tr>
<tr>
<td>initialize_final_statistics</td>
<td>625</td>
</tr>
<tr>
<td>initialize_random_variable_parameters</td>
<td>626</td>
</tr>
<tr>
<td>initialize_random_variable_types</td>
<td>625</td>
</tr>
<tr>
<td>initialize_random_variables</td>
<td>624</td>
</tr>
<tr>
<td>initialize_run</td>
<td>624</td>
</tr>
<tr>
<td>print_distribution_map</td>
<td>627</td>
</tr>
<tr>
<td>print_distribution_mappings</td>
<td>626</td>
</tr>
<tr>
<td>run</td>
<td>625</td>
</tr>
<tr>
<td>set_u_to_x_mapping</td>
<td>627</td>
</tr>
<tr>
<td>vars_u_to_x_mapping</td>
<td>626</td>
</tr>
<tr>
<td>vars_x_to_u_mapping</td>
<td>627</td>
</tr>
<tr>
<td>Dakota::NonDAIImpSampling</td>
<td>628</td>
</tr>
<tr>
<td>initialize</td>
<td>631</td>
</tr>
<tr>
<td>NonDAIImpSampling</td>
<td>630</td>
</tr>
<tr>
<td>Dakota::NonDAdaptiveSampling</td>
<td>632</td>
</tr>
<tr>
<td>~NonDAdaptiveSampling</td>
<td>637</td>
</tr>
<tr>
<td>NonDAdaptiveSampling</td>
<td>637</td>
</tr>
<tr>
<td>Dakota::NonDBayesCalibration</td>
<td>638</td>
</tr>
<tr>
<td>NonDBayesCalibration</td>
<td>639</td>
</tr>
<tr>
<td>Dakota::NonDCalibration</td>
<td>640</td>
</tr>
<tr>
<td>NonDCalibration</td>
<td>642</td>
</tr>
<tr>
<td>Dakota::NonDCubature</td>
<td>643</td>
</tr>
<tr>
<td>increment_grid_preference</td>
<td>645</td>
</tr>
<tr>
<td>increment_reference</td>
<td>645</td>
</tr>
<tr>
<td>NonDCubature</td>
<td>644</td>
</tr>
<tr>
<td>num_samples</td>
<td>645</td>
</tr>
<tr>
<td>sampling_reset</td>
<td>645</td>
</tr>
<tr>
<td>Dakota::NonDDREAMBayesCalibration</td>
<td>646</td>
</tr>
<tr>
<td>likelihoodScale</td>
<td>650</td>
</tr>
<tr>
<td>NonDDREAMBayesCalibration</td>
<td>648</td>
</tr>
<tr>
<td>prior_density</td>
<td>649</td>
</tr>
<tr>
<td>prior_sample</td>
<td>649</td>
</tr>
<tr>
<td>problem_size</td>
<td>649</td>
</tr>
<tr>
<td>quantify_uncertainty</td>
<td>649</td>
</tr>
<tr>
<td>sample_likelihood</td>
<td>649</td>
</tr>
<tr>
<td>Dakota::NonDExpansion</td>
<td>651</td>
</tr>
<tr>
<td>compute_covariance_metric</td>
<td>656</td>
</tr>
<tr>
<td>compute_final_statistics_metric</td>
<td>657</td>
</tr>
<tr>
<td>compute_statistics</td>
<td>657</td>
</tr>
<tr>
<td>increment_specification_sequence</td>
<td>656</td>
</tr>
<tr>
<td>useDerivs</td>
<td>658</td>
</tr>
<tr>
<td>Dakota::NonDGlobalEvidence</td>
<td>659</td>
</tr>
<tr>
<td>Dakota::NonDGlobalInterval</td>
<td>661</td>
</tr>
<tr>
<td>Dakota::NonDGlobalReliability</td>
<td>665</td>
</tr>
<tr>
<td>Dakota::NonDGlobalSingleInterval</td>
<td>668</td>
</tr>
<tr>
<td>Dakota::NonDGPImpSampling</td>
<td>670</td>
</tr>
<tr>
<td>~NonDGPImpSampling</td>
<td>672</td>
</tr>
<tr>
<td>NonDGPImpSampling</td>
<td>672</td>
</tr>
<tr>
<td>quantify_uncertainty</td>
<td>673</td>
</tr>
<tr>
<td>Dakota::NonDGPMSABayesCalibration</td>
<td>674</td>
</tr>
<tr>
<td>NonDGPMSABayesCalibration</td>
<td>676</td>
</tr>
<tr>
<td>quantify_uncertainty</td>
<td>676</td>
</tr>
<tr>
<td>randomSeed</td>
<td>677</td>
</tr>
<tr>
<td>Dakota::NonDIncremLHSSampling</td>
<td>678</td>
</tr>
<tr>
<td>NonDIncremLHSSampling</td>
<td>679</td>
</tr>
<tr>
<td>quantify_uncertainty</td>
<td>679</td>
</tr>
<tr>
<td>Dakota::NonDIntegration</td>
<td>681</td>
</tr>
</tbody>
</table>
anisotropic_order_to_dimension_preference, 683
dimension_preference_to_anisotropic_order, 683
check_variables, 683
NonDIntegration, 683
Dakota::NonDInterval, 685
print_results, 687
Dakota::NonDLHSEvidence, 688
Dakota::NonDLSInterval, 690
Dakota::NonDLHSSampling, 692
NonDLHSSampling, 693
quantify_uncertainty, 694
Dakota::NonDLHSSingleInterval, 695
Dakota::NonDLocalEvidence, 697
Dakota::NonDLocalInterval, 699
Dakota::NonDLocalReliability, 702
dg_ds_eval, 712
dp2_dbeta_factor, 712
initial_taylor_series, 709
initialize_class_data, 710
initialize_level_data, 710
initialize_mpp_search_data, 710
PMA2_constraint_eval, 709
PMA_constraint_eval, 709
PMA_objective_eval, 708
probability, 712
RIA_constraint_eval, 708
RIA_objective_eval, 708
update_level_data, 711
update_mpp_search_data, 711
Dakota::NonDLocalSingleInterval, 714
Dakota::NonDPOFDarts, 716
Dakota::NonDPolynomialChaos, 719
increment_order, 722
increment_specification_sequence, 722
NonDPolynomialChaos, 721, 722
Dakota::NonDQuadrature, 724
initialize_grid, 727
NonDQuadrature, 727
num_samples, 728
sampling_reset, 728
Dakota::NonDQUESOBayesCalibration, 729
NonDQUESOBayesCalibration, 731
quantify_uncertainty, 731
Dakota::NonDReliability, 732
Dakota::NonDSampling, 734
get_parameter_sets, 739, 740
NonDSampling, 738
num_samples, 739
sampling_reset, 739
view_aleatory_uncertain_counts, 740
view_design_counts, 740
view_epistemic_uncertain_counts, 740
view_uncertain_counts, 741
Dakota::NonDSparseGrid, 742
NonDSparseGrid, 744
num_samples, 745
sampling_reset, 745
Dakota::NonDStochCollocation, 746
compute_covariance_metric, 748
compute_final_statistics_metric, 748
NonDStochCollocation, 747
Dakota::NonlinearCGOptimizer, 749
brent_minimize, 751
Dakota::NPSOLOptimizer, 753
NPSOLOptimizer, 755
Dakota::Optimizer, 757
finalize_run, 760
initialize_run, 760
local_objective_recast_retrieve, 762
objective_reduction, 761
Optimizer, 759
post_run, 760
primary_resp_reducer, 761
print_results, 760
reduce_model, 761
run, 760
Dakota::ParallelConfiguration, 763
Dakota::ParallelLevel, 767
Dakota::ParallelLibrary, 771
close_streams, 783
increment_parallel_configuration, 783
init_communicators, 784
init_mpi_comm, 784
manage_outputs_restart, 783
ParallelLibrary, 781, 782
resolve_inputs, 784
specify_outputs_restart, 782
split_filenames, 784
Dakota::ParamResponsePair, 786
evalInterfaceIds, 789
ParamResponsePair, 788
read, 788
write, 788
Dakota::ParamStudy, 790
load_distribute_points, 794
post_run, 794
pre_run, 794
Dakota::partial_prp_equality, 796
Dakota::partial_prp_hash, 797
Dakota::PecosApproximation, 798
    build, 804
    finalize, 805
    pop, 805
    rebuild, 804
    restore, 805
Dakota::ProblemDescDB, 806
    ~ProblemDescDB, 815
    get_db, 817
    manage_inputs, 816
    operator=, 816
    parse_inputs, 816
    post_process, 816
ProblemDescDB, 815
Dakota::ProcessApplicInterface, 818
    synchronous_local_analyses, 821
Dakota::ProcessHandleApplicInterface, 823
    asynchronous_local_analyses, 826
    check_wait, 826
    create_evaluation_process, 825
    init_communicators_checks, 825
    serve_analyses_asynch, 826
    set_communicators_checks, 825
    synchronous_local_analysis, 825
Dakota::PStudyDACE, 828
    print_results, 829
    run, 829
    volumetric_quality, 830
Dakota::PSUADEDesignCompExp, 831
    enforce_input_rules, 834
    num_samples, 833
    post_run, 833
    pre_run, 833
PSUADEDesignCompExp, 833
Dakota::PythonInterface, 835
    derived_map_ac, 836
    python_convert_int, 836
Dakota::RecastBaseConstructor, 838
Dakota::RecastModel, 839
    derived_compute_response, 847
    eval_tag_prefix, 847
    initialize, 847
RecastModel, 846
    update_from_sub_model, 848
Dakota::RelaxedVarConstraints, 850
    RelaxedVarConstraints, 851
    reshape, 851
Dakota::RelaxedVariables, 852
    read_tabular, 853
    RelaxedVariables, 853
Dakota::Response, 854
    Response, 858
Dakota::ResponseRep, 860
    functionGradients, 866
    read, 863–865
    read_annotated, 863
    read_tabular, 864
    reset, 866
    reset_inactive, 866
    reshape, 866
    ResponseRep, 862, 863
    update, 865
    update_partial, 865
    write, 863–865
    write_annotated, 864
    write_tabular, 864
Dakota::ResultsDBAny, 868
    add_data, 870
    array_insert, 870
    extract_data, 870
Dakota::ResultsEntry, 871
Dakota::ResultsID, 873
Dakota::ResultsManager, 875
Dakota::ResultsNames, 878
Dakota::RichExtrapVerification, 880
    converge_order, 882
    converge_qoi, 882
    estimate_order, 882
    print_results, 881
Dakota::ScilabInterface, 883
Dakota::SensAnalysisGlobal, 885
Dakota::SequentialHybridStrategy, 887
    extract_parameter_sets, 890
    pack_parameters_buffer, 889
    pack_results_buffer, 889
    run_sequential, 890
    run_sequential_adaptive, 890
    unpack_parameters_buffer, 889
    unpack_results_buffer, 890
Dakota::SharedVariablesData, 894
Dakota::SharedVariablesDataRep, 899
    allContinuousIds, 902
    SharedVariablesDataRep, 902
Dakota::SingleMethodStrategy, 903
Dakota::SingleModel, 905
    eval_tag_prefix, 907
Dakota::SNLLBase, 908
Dakota::SNLLEastSq, 911
    constraint1_evaluator_gn, 914
    constraint2_evaluator_gn, 914
    nlf2_evaluator_gn, 913
    post_run, 913
Dakota::SNLLOptimizer, 916
    constraint0_evaluator, 922
    constraint1_evaluator, 922
    constraint2_evaluator, 922
    nlf0_evaluator, 921
    nlf1_evaluator, 921
    nlf2_evaluator, 922
    SNLLOptimizer, 920
Dakota::SOLBase, 924
Dakota::SpawnApplicInterface, 927
Dakota::Strategy, 929
    ~Strategy, 933
    free_iterator, 936
    get_strategy, 936
    init_iterator, 935
    init_iterator_parallelism, 935
    operator=, 934
    pack_parameters_buffer, 934
    pack_results_buffer, 934
    run_iterator, 935
    schedule_iterators, 936
    self_schedule_iterators, 936
    serve_iterators, 936
    Strategy, 933
    unpack_parameters_buffer, 934
    unpack_results_buffer, 934
Dakota::SurfpackApproximation, 938
    add_anchor_to_surfdata, 942
    build, 941
    hessian, 941
    SurfpackApproximation, 941
    surrogates_to_surf_data, 941
Dakota::SurrBasedGlobalMinimizer, 943
Dakota::SurrBasedLocalMinimizer, 945
    approx_subprob_constraint_eval, 952
    approx_subprob_objective_eval, 952
    hard_convergence_check, 950
    hom_constraint_eval, 952
    hom_objective_eval, 952
    minimize_surrogates, 950
    tr_ratio_check, 951
    update_penalty, 951
Dakota::SurrBasedMinimizer, 954
    augmented_lagrangian_merit, 958
    constraint_violation, 958
    lagrangian_merit, 958
    penalty_merit, 958
    print_results, 956
    run, 956
    update_augmented_lagrange_multipliers, 957
    update_filter, 957
    update_lagrange_multipliers, 957
Dakota::SurrogModel, 960
    approxBuilds, 963
    force_rebuild, 963
    responseMode, 963
Dakota::SysCallApplicInterface, 965
    init_communicators_checks, 967
    set_communicators_checks, 967
    spawn_analysis_to_shell, 968
    spawn_evaluation_to_shell, 967
    spawn_input_filter_to_shell, 968
    spawn_output_filter_to_shell, 968
    synchronous_local_analysis, 967
    test_local_evaluations, 966
    wait_local_evaluations, 966
Dakota::TANA3Approximation, 970
    build, 971
    clear_current, 971
Dakota::TaylorApproximation, 973
    build, 974
Dakota::TestDriverInterface, 975
    derived_map_ac, 978
    herbie, 979
    herbie1D, 978
    mc_api_run, 979
    separable_combine, 979
    shubert1D, 979
    smooth_herbie, 979
    smooth_herbie1D, 978
Dakota::TrackerHTTP, 981
    send_data_using_get, 983
    send_data_using_post, 983
Dakota::Variables, 984
    ~Variables, 994
    build_views, 995
    copy, 995
    get_variables, 996
    method_map, 996
    operator=, 995
    Variables, 994, 995
Dakota::Verification, 997
    print_results, 998
run, 998
dakota_dll_api.cpp, 999
dakota_stop, 1000
dakota_dll_api.h, 1001
dakota_stop, 1002
dakota_response_to_colin_response
    Dakota::COLINApplication, 297
dakota_stop
    dakota_dll_api.cpp, 1000
    dakota_dll_api.h, 1002
dakota_tabular_io.hpp, 1003
data_transform_model
    Dakota::Minimizer, 530
DAUIVLbl
    Dakota, 229
DAURVLbl
    Dakota, 229
DDACEDesignCompExp
    Dakota::DDACEDesignCompExp, 405
derivative_concurrency
    Dakota::Model, 571
derived_async_compute_response
    Dakota::DataFitSurrModel, 340
    Dakota::HierarchSurrModel, 477
    Dakota::NestedModel, 590
derived_compute_response
    Dakota::DataFitSurrModel, 340
    Dakota::HierarchSurrModel, 477
    Dakota::NestedModel, 590
    Dakota::RecastModel, 847
derived_init_communicators
    Dakota::DataFitSurrModel, 344
    Dakota::NestedModel, 591
derived_map_ac
    Dakota::DirectApplicInterface, 412
    Dakota::MatlabInterface, 521
    Dakota::PythonInterface, 836
    Dakota::TestDriverInterface, 978
derived_master_overload
    Dakota::NestedModel, 591
derived_parse_inputs
    Dakota::NIDRProblemDescDB, 598
derived_synchronize
    Dakota::DataFitSurrModel, 340
    Dakota::HierarchSurrModel, 477
derived_synchronize_nowait
    Dakota::DataFitSurrModel, 341
    Dakota::HierarchSurrModel, 478
derivVarsVector
    Dakota::ActiveSet, 237
Description
    Dakota::JEGAOptimizer::Evaluator, 440
DestroyAlgorithm
    Dakota::JEGAOptimizer::Driver, 425
DEUIVLbl
    Dakota, 229
DEURVLbl
    Dakota, 230
DF
    Dakota::CONMINOptimizer, 320
dg_ds_eval
    Dakota::NonDLocalReliability, 712
dimension_preference_to_anisotropic_order
    Dakota::NonDIntegration, 683
DiscSetLbl
    Dakota, 230
div_index_map
    Dakota::NestedModel, 592
dotFDSinfo
    Dakota::DOTOptimizer, 421
dotInfo
    Dakota::DOTOptimizer, 421
dotMethod
    Dakota::DOTOptimizer, 421
dp2_dbeta_factor
    Dakota::NonDLocalReliability, 712
Driver
    Dakota::JEGAOptimizer::Driver, 424
drv_index_map
    Dakota::NestedModel, 592
duplication_detect
    Dakota::ApplicationInterface, 257
enforce_input_rules
    Dakota::FSUDesignCompExp, 454
    Dakota::PSUADEDesignCompExp, 834
enroll
    Dakota::GetLongOpt, 465
estimate_derivatives
    Dakota::Model, 572
estimate_message_lengths
    Dakota::Model, 570
estimate_order
    Dakota::RichExtrapVerification, 882
eval_tag_prefix
    Dakota::Interface, 490
    Dakota::Iterator, 504
    Dakota::Model, 569
    Dakota::RecastModel, 847
find_optimum
    Dakota::APPSOptimizer, 285
    Dakota::COLINOptimizer, 300
    Dakota::JEGAOptimizer, 513
flush
    Dakota, 131
    Dakota::CommandShell, 309
force_rebuild
    Dakota::SurrogateModel, 963
fpinit_ASL
    main.cpp, 1010
free_iterator
    Dakota::Strategy, 936
FSUDesignCompExp
    Dakota::FSUDesignCompExp, 453
functionGradients
    Dakota::ResponseRep, 866
functionSurfaces
    Dakota::ApproximationInterface, 278
G1
    Dakota::CONMINOptimizer, 319
G2
    Dakota::CONMINOptimizer, 319
GaussProcApproximation
    Dakota::GaussProcApproximation, 461
get_approx
    Dakota::Approximation, 271
get_best_sample
    Dakota::EffGlobalMinimizer, 428
get_confidence_intervals
    Dakota::LeastSq, 518
get_constraints
    Dakota::Constraints, 332
get_cwd
    Dakota, 132
get_db
    Dakota::ProblemDescDB, 817
get_interface
    Dakota::Interface, 491
get_iterator
    Dakota::Iterator, 505
get_model
    Dakota::Model, 572
get_npath
    Dakota::Model, 572
get_npath
    Dakota::NonDSampling, 739, 740
get_strategy
    Dakota::Strategy, 936
<table>
<thead>
<tr>
<th>Function</th>
<th>Class/Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_variables</td>
<td>Dakota::Variables, 996</td>
</tr>
<tr>
<td>GetBestMOSolutions</td>
<td>Dakota::JEGAOptimizer, 512</td>
</tr>
<tr>
<td>GetBestSolutions</td>
<td>Dakota::JEGAOptimizer, 512</td>
</tr>
<tr>
<td>GetBestSOSolutions</td>
<td>Dakota::JEGAOptimizer, 512</td>
</tr>
<tr>
<td>GetDescription</td>
<td>Dakota::JEGAOptimizer::Evaluator, 443</td>
</tr>
<tr>
<td>getdist</td>
<td>Dakota, 132</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>Dakota::GetLongOpt, 465</td>
</tr>
<tr>
<td>GetName</td>
<td>Dakota::JEGAOptimizer::Evaluator, 443</td>
</tr>
<tr>
<td>GetNumberLinearConstraints</td>
<td>Dakota::JEGAOptimizer::Evaluator, 442</td>
</tr>
<tr>
<td>GetNumberNonLinearConstraints</td>
<td>Dakota::JEGAOptimizer::Evaluator, 441</td>
</tr>
<tr>
<td>getRmax</td>
<td>Dakota, 133</td>
</tr>
<tr>
<td>gnewton_set_recast</td>
<td>Dakota::Minimizer, 531</td>
</tr>
<tr>
<td>GPmodel_apply</td>
<td>Dakota::GaussProcApproximation, 461</td>
</tr>
<tr>
<td>hard_convergence_check</td>
<td>Dakota::SurrBasedLocalMinimizer, 950</td>
</tr>
<tr>
<td>herbie</td>
<td>Dakota::TestDriverInterface, 979</td>
</tr>
<tr>
<td>herbie1D</td>
<td>Dakota::TestDriverInterface, 978</td>
</tr>
<tr>
<td>hessian</td>
<td>Dakota::SurfpackApproximation, 941</td>
</tr>
<tr>
<td>hom_constraint_eval</td>
<td>Dakota::SurrBasedLocalMinimizer, 952</td>
</tr>
<tr>
<td>hom_objective_eval</td>
<td>Dakota::SurrBasedLocalMinimizer, 952</td>
</tr>
<tr>
<td>IC</td>
<td>Dakota::CONMINOptimizer, 320</td>
</tr>
<tr>
<td>id_vars_exact_compare</td>
<td>Dakota, 134</td>
</tr>
<tr>
<td>import_points</td>
<td>Dakota::DataFitSurrModel, 344</td>
</tr>
<tr>
<td>increment_grid_preference</td>
<td>Dakota::NonDCubature, 645</td>
</tr>
<tr>
<td>increment_order</td>
<td>Dakota::NonDPolynomialChaos, 722</td>
</tr>
<tr>
<td>increment_parallel_configuration</td>
<td>Dakota::ParallelLibrary, 783</td>
</tr>
<tr>
<td>increment_reference</td>
<td>Dakota::NonDCubature, 645</td>
</tr>
<tr>
<td>increment_specification_sequence</td>
<td>Dakota::NonDExpansion, 656</td>
</tr>
<tr>
<td>init_communicators</td>
<td>Dakota::Model, 570</td>
</tr>
<tr>
<td>init_communicators_checks</td>
<td>Dakota::ParallelLibrary, 784</td>
</tr>
<tr>
<td>init_iterator</td>
<td>Dakota::Strategy, 935</td>
</tr>
<tr>
<td>init_iterator_parallelism</td>
<td>Dakota::Strategy, 935</td>
</tr>
<tr>
<td>init_mpi_comm</td>
<td>Dakota::ParallelLibrary, 784</td>
</tr>
<tr>
<td>init_serial</td>
<td>Dakota::ApplicationInterface, 254</td>
</tr>
<tr>
<td>initial_points</td>
<td>Dakota::Model, 570</td>
</tr>
<tr>
<td>initial_taylor_series</td>
<td>Dakota::NonDLocalReliability, 709</td>
</tr>
<tr>
<td>initialize</td>
<td>Dakota::NonDAadaptImpSampling, 631</td>
</tr>
<tr>
<td>initialize_class_data</td>
<td>Dakota::RecastModel, 847</td>
</tr>
<tr>
<td>initialize_final_statistics</td>
<td>Dakota::NonD, 625</td>
</tr>
<tr>
<td>initialize_graphics</td>
<td>Dakota::Iterator, 502</td>
</tr>
<tr>
<td>initialize_grid</td>
<td>Dakota::NonDQuadrature, 727</td>
</tr>
<tr>
<td>initialize_h</td>
<td>Dakota::Model, 572</td>
</tr>
<tr>
<td>initialize_level_data</td>
<td>Dakota::NonDLocalReliability, 710</td>
</tr>
<tr>
<td>initialize_mpp_search_data</td>
<td>Dakota::NonDLocalReliability, 710</td>
</tr>
<tr>
<td>initialize_random_variable_parameters</td>
<td>Dakota::NonD, 626</td>
</tr>
<tr>
<td>initialize_random_variable_types</td>
<td>Dakota::NonD, 625</td>
</tr>
<tr>
<td>Keyword</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>kw_129</td>
<td>170</td>
</tr>
<tr>
<td>kw_13</td>
<td>141</td>
</tr>
<tr>
<td>kw_130</td>
<td>170</td>
</tr>
<tr>
<td>kw_131</td>
<td>170</td>
</tr>
<tr>
<td>kw_132</td>
<td>171</td>
</tr>
<tr>
<td>kw_133</td>
<td>171</td>
</tr>
<tr>
<td>kw_134</td>
<td>171</td>
</tr>
<tr>
<td>kw_135</td>
<td>171</td>
</tr>
<tr>
<td>kw_136</td>
<td>171</td>
</tr>
<tr>
<td>kw_137</td>
<td>172</td>
</tr>
<tr>
<td>kw_138</td>
<td>172</td>
</tr>
<tr>
<td>kw_139</td>
<td>173</td>
</tr>
<tr>
<td>kw_14</td>
<td>141</td>
</tr>
<tr>
<td>kw_140</td>
<td>173</td>
</tr>
<tr>
<td>kw_141</td>
<td>173</td>
</tr>
<tr>
<td>kw_142</td>
<td>173</td>
</tr>
<tr>
<td>kw_143</td>
<td>173</td>
</tr>
<tr>
<td>kw_144</td>
<td>174</td>
</tr>
<tr>
<td>kw_145</td>
<td>174</td>
</tr>
<tr>
<td>kw_146</td>
<td>174</td>
</tr>
<tr>
<td>kw_147</td>
<td>174</td>
</tr>
<tr>
<td>kw_148</td>
<td>174</td>
</tr>
<tr>
<td>kw_149</td>
<td>175</td>
</tr>
<tr>
<td>kw_15</td>
<td>141</td>
</tr>
<tr>
<td>kw_150</td>
<td>170</td>
</tr>
<tr>
<td>kw_151</td>
<td>171</td>
</tr>
<tr>
<td>kw_152</td>
<td>172</td>
</tr>
<tr>
<td>kw_153</td>
<td>172</td>
</tr>
<tr>
<td>kw_154</td>
<td>173</td>
</tr>
<tr>
<td>kw_155</td>
<td>173</td>
</tr>
<tr>
<td>kw_156</td>
<td>173</td>
</tr>
<tr>
<td>kw_157</td>
<td>173</td>
</tr>
<tr>
<td>kw_158</td>
<td>173</td>
</tr>
<tr>
<td>kw_159</td>
<td>173</td>
</tr>
<tr>
<td>kw_16</td>
<td>173</td>
</tr>
<tr>
<td>kw_160</td>
<td>174</td>
</tr>
<tr>
<td>kw_161</td>
<td>174</td>
</tr>
<tr>
<td>kw_162</td>
<td>174</td>
</tr>
<tr>
<td>kw_163</td>
<td>174</td>
</tr>
<tr>
<td>kw_164</td>
<td>174</td>
</tr>
<tr>
<td>kw_165</td>
<td>174</td>
</tr>
<tr>
<td>kw_166</td>
<td>174</td>
</tr>
<tr>
<td>kw_167</td>
<td>174</td>
</tr>
<tr>
<td>kw_168</td>
<td>174</td>
</tr>
<tr>
<td>kw_169</td>
<td>174</td>
</tr>
<tr>
<td>kw_17</td>
<td>174</td>
</tr>
<tr>
<td>kw_170</td>
<td>174</td>
</tr>
<tr>
<td>kw_171</td>
<td>174</td>
</tr>
<tr>
<td>kw_172</td>
<td>174</td>
</tr>
</tbody>
</table>
Dakota, 181
kw_173
Dakota, 181
kw_174
Dakota, 181
kw_175
Dakota, 181
kw_176
Dakota, 182
kw_177
Dakota, 182
kw_178
Dakota, 182
kw_179
Dakota, 182
kw_18
Dakota, 182
kw_180
Dakota, 182
kw_181
Dakota, 182
kw_182
Dakota, 183
kw_183
Dakota, 184
kw_184
Dakota, 184
kw_185
Dakota, 184
kw_186
Dakota, 185
kw_187
Dakota, 185
kw_188
Dakota, 185
kw_189
Dakota, 185
kw_19
Dakota, 185
kw_190
Dakota, 185
kw_191
Dakota, 186
kw_192
Dakota, 186
kw_193
Dakota, 186
kw_194
Dakota, 186
kw_195
Dakota, 187
kw_196
Dakota, 187
kw_197
Dakota, 188
kw_198
Dakota, 188
kw_199
Dakota, 189
kw_2
Dakota, 138
kw_20
Dakota, 142
kw_200
Dakota, 142
kw_201
Dakota, 189
kw_202
Dakota, 189
kw_203
Dakota, 190
kw_204
Dakota, 190
kw_205
Dakota, 190
kw_206
Dakota, 190
kw_207
Dakota, 191
kw_208
Dakota, 191
kw_209
Dakota, 191
kw_21
Dakota, 191
kw_210
Dakota, 192
kw_211
Dakota, 192
kw_212
Dakota, 193
kw_213
Dakota, 193
kw_214
Dakota, 193
kw_215
Dakota, 193
kw_216
Dakota, 194
| kw_217  | Dakota, 194 | kw_24  | Dakota, 143 |
| kw_218  | Dakota, 194 | kw_240 | Dakota, 200 |
| kw_219  | Dakota, 195 | kw_241 | Dakota, 200 |
| kw_22   | Dakota, 143 | kw_242 | Dakota, 200 |
| kw_221  | Dakota, 195 | kw_243 | Dakota, 200 |
| kw_222  | Dakota, 196 | kw_244 | Dakota, 201 |
| kw_223  | Dakota, 196 | kw_245 | Dakota, 201 |
| kw_224  | Dakota, 196 | kw_246 | Dakota, 202 |
| kw_225  | Dakota, 196 | kw_247 | Dakota, 202 |
| kw_226  | Dakota, 196 | kw_248 | Dakota, 202 |
| kw_227  | Dakota, 197 | kw_249 | Dakota, 202 |
| kw_228  | Dakota, 197 | kw_25  | Dakota, 202 |
| kw_229  | Dakota, 197 | kw_250 | Dakota, 143 |
| kw_23   | Dakota, 143 | kw_251 | Dakota, 203 |
| kw_230  | Dakota, 197 | kw_252 | Dakota, 203 |
| kw_231  | Dakota, 198 | kw_253 | Dakota, 204 |
| kw_232  | Dakota, 198 | kw_254 | Dakota, 204 |
| kw_233  | Dakota, 198 | kw_255 | Dakota, 205 |
| kw_234  | Dakota, 198 | kw_256 | Dakota, 205 |
| kw_235  | Dakota, 199 | kw_257 | Dakota, 205 |
| kw_236  | Dakota, 199 | kw_258 | Dakota, 205 |
| kw_237  | Dakota, 199 | kw_259 | Dakota, 205 |
| kw_238  | Dakota, 199 | kw_26  | Dakota, 206 |
| kw_239  | Dakota, 199 | kw_260 | Dakota, 206 |
| kw_240  | Dakota, 199 | kw_261 | Dakota, 206 |
Dakota, 206
kw_262
Dakota, 207
kw_263
Dakota, 207
kw_264
Dakota, 208
kw_265
Dakota, 208
kw_266
Dakota, 208
kw_267
Dakota, 208
kw_268
Dakota, 209
kw_269
Dakota, 209
kw_27
Dakota, 144
kw_270
Dakota, 209
kw_271
Dakota, 210
kw_272
Dakota, 210
kw_273
Dakota, 210
kw_274
Dakota, 210
kw_275
Dakota, 211
kw_276
Dakota, 211
kw_277
Dakota, 211
kw_278
Dakota, 211
kw_279
Dakota, 211
kw_28
Dakota, 144
kw_280
Dakota, 212
kw_281
Dakota, 212
kw_282
Dakota, 212
kw_283
Dakota, 213
kw_284
Dakota, 213
kw_285
Dakota, 214
kw_286
Dakota, 214
kw_287
Dakota, 214
kw_288
Dakota, 215
kw_289
Dakota, 215
kw_29
Dakota, 216
kw_290
Dakota, 216
kw_291
Dakota, 216
kw_292
Dakota, 216
kw_293
Dakota, 216
kw_294
Dakota, 217
kw_295
Dakota, 217
kw_296
Dakota, 217
kw_297
Dakota, 217
kw_298
Dakota, 217
kw_299
Dakota, 217
kw_3
Dakota, 218
kw_3
Dakota, 218
kw_3
Dakota, 218
kw_3
Dakota, 218
kw_3
Dakota, 218
kw_3
Dakota, 138
kw_30
Dakota, 145
kw_300
Dakota, 219
kw_301
Dakota, 219
kw_302
Dakota, 219
kw_303
Dakota, 219
kw_304
Dakota, 220
kw_305
Dakota, 220
Dakota, 220
kw_306
Dakota, 221
kw_307
Dakota, 221
kw_308
Dakota, 221
kw_309
Dakota, 222
kw_31
Dakota, 145
kw_310
Dakota, 222
kw_311
Dakota, 222
kw_312
Dakota, 223
kw_313
Dakota, 223
kw_315
Dakota, 224
kw_316
Dakota, 224
kw_317
Dakota, 224
kw_318
Dakota, 225
kw_319
Dakota, 225
kw_32
Dakota, 146
kw_320
Dakota, 225
kw_321
Dakota, 225
kw_322
Dakota, 226
kw_323
Dakota, 226
kw_324
Dakota, 227
kw_325
Dakota, 227
kw_326
Dakota, 227
kw_327
Dakota, 228
kw_328
Dakota, 228
kw_33
Dakota, 146
kw_330
Dakota, 228
kw_34
Dakota, 146
kw_35
Dakota, 146
kw_36
Dakota, 146
kw_37
Dakota, 147
kw_38
Dakota, 147
kw_39
Dakota, 147
kw_4
Dakota, 138
kw_40
Dakota, 147
kw_41
Dakota, 147
kw_42
Dakota, 148
kw_43
Dakota, 148
kw_44
Dakota, 148
kw_45
Dakota, 148
kw_46
Dakota, 149
kw_47
Dakota, 149
kw_48
Dakota, 149
kw_49
Dakota, 149
kw_5
Dakota, 149
kw_50
Dakota, 138
kw_51
Dakota, 149
kw_52
Dakota, 150
kw_53
Dakota, 150
kw_54
Dakota, 163

lagrangian_merit
  Dakota::SurrBasedMinimizer, 958

LeastSq
  Dakota::LeastSq, 517

library_mode.cpp, 1007
  main, 1008
  model_interface_plugins, 1008
  my_callback_function, 1008
  run_dakota_mixed, 1007
  run_dakota_parse, 1007

library_split.cpp, 1009

likelihoodScale
  Dakota::NonDDREAMBayesCalibration, 650

lin_coeffs_modify_n2s
  Dakota::Minimizer, 535

load_distribute_points
  Dakota::ParamStudy, 794

load_parameters
  Dakota::NomadOptimizer, 615

load_scalar
  Dakota::ExperimentData, 447

LoadAlgorithmConfig
  Dakota::JEGAOptimizer, 510

LoadDakotaResponses
  Dakota::JEGAOptimizer, 510

LoadProblemConfig
  Dakota::JEGAOptimizer, 510

LoadTheConstraints
  Dakota::JEGAOptimizer, 511

LoadTheDesignVariables
  Dakota::JEGAOptimizer, 511

LoadTheObjectiveFunctions
  Dakota::JEGAOptimizer, 511

LoadTheParameterDatabase
  Dakota::JEGAOptimizer, 510

local_eval_concurrency
  Dakota::Model, 568

local_eval_synchronization
  Dakota::Model, 568

local_objective_recast_retrieve
  Dakota::Optimizer, 762

lookup_by_val
  Dakota, 134

main
  library_mode.cpp, 1008
  main.cpp, 1010
  fpinit_ASL, 1010
  main, 1010

manage_asv
  Dakota::Model, 574

manage_inputs
  Dakota::ProblemDescDB, 816

manage_linear_constraints
  Dakota::Constraints, 332

manage_outputs_restart
  Dakota::ParallelLibrary, 783

MandatoryValue
  Dakota::GetLongOpt, 465

map
  Dakota::ApplicationInterface, 254

map_domain
  Dakota::COLINAApplication, 297

map_xi_to_x
  Dakota::EfficientSubspaceMethod, 432

master_dynamic_schedule_analyses
  Dakota::ApplicationInterface, 256

master_dynamic_schedule_evaluations
  Dakota::ApplicationInterface, 257

master_dynamic_schedule_evaluations_nowait
  Dakota::ApplicationInterface, 259

matlab_engine_run
  Dakota::MatlabInterface, 521

mc_api_run
  Dakota::TestDriverInterface, 979

method_map
  Dakota::Variables, 996

mindist
  Dakota, 132

mindistindx
  Dakota, 132

minimize_residuals
  Dakota::NL2SOLLeastSq, 603

minimize_surrogates
  Dakota::SurrBasedLocalMinimizer, 950

Minimizer
  Dakota::Minimizer, 529

MixedVarConstraints
  Dakota::MixedVarConstraints, 537

MixedVariables
  Dakota::MixedVariables, 539

Model
  Dakota::Model, 565, 566

model_interface_plugins
  library_mode.cpp, 1008
modify_n2s
  Dakota::Minimizer, 535
modify_s2n
  Dakota::Minimizer, 531
MS1
  Dakota::CONMINOptimizer, 319
my_callback_function
  library_mode.cpp, 1008
my_cp
  Dakota, 132
N1
  Dakota::CONMINOptimizer, 318
N2
  Dakota::CONMINOptimizer, 318
N3
  Dakota::CONMINOptimizer, 318
N4
  Dakota::CONMINOptimizer, 318
N5
  Dakota::CONMINOptimizer, 318
Name
  Dakota::JEGAOptimizer::Evaluator, 440
NCSUOptimizer
  Dakota::NCSUOptimizer, 583
need_resp_trans_byvars
  Dakota::Minimizer, 531
new_dataset
  Dakota::Graphics, 469
nlf0_evaluator
  Dakota::SNLLOptimizer, 921
nlf1_evaluator
  Dakota::SNLLOptimizer, 921
nlf2_evaluator
  Dakota::SNLLOptimizer, 922
nlf2_evaluator_gn
  Dakota::SNLLEastSq, 913
NLSSOLLeastSq
  Dakota::NLSSOLLeastSq, 611
NomadOptimizer
  Dakota::NomadOptimizer, 615
NonDAdaptImpSampling
  Dakota::NonDAdaptImpSampling, 630
NonDAdaptiveSampling
  Dakota::NonDAdaptiveSampling, 637
NonDBayesCalibration
  Dakota::NonDBayesCalibration, 639
NonDCalibration
  Dakota::NonDCalibration, 642
NonDCubature
  Dakota::NonDCubature, 644
NonDDREAMBayesCalibration
  Dakota::NonDDREAMBayesCalibration, 648
NonDGPImpSampling
  Dakota::NonDGPImpSampling, 672
NonDGPMASBayesCalibration
  Dakota::NonDGPMASBayesCalibration, 676
NonDIncLHSSampling
  Dakota::NonDIncLHSSampling, 679
NonDIntegration
  Dakota::NonDIntegration, 683
NonDLHSSampling
  Dakota::NonDLHSSampling, 693
NonDPolynomialChaos
  Dakota::NonDPolynomialChaos, 721, 722
NonDQuadrature
  Dakota::NonDQuadrature, 727
NonDQUESOBayesCalibration
  Dakota::NonDQUESOBayesCalibration, 731
NonDSampling
  Dakota::NonDSampling, 738
NonDSparseGrid
  Dakota::NonDSparseGrid, 744
NonDStochCollocation
  Dakota::NonDStochCollocation, 747
NPSOLOptimizer
  Dakota::NPSOLOptimizer, 755
num_samples
  Dakota::DDACEDesignCompExp, 406
  Dakota::FSUDesignCompExp, 454
  Dakota::Iterator, 503
  Dakota::NonDCubature, 645
  Dakota::NonDQuadrature, 728
  Dakota::NonDSampling, 739
  Dakota::NonDStochCollocation, 745
  Dakota::PSUADEDesignCompExp, 833
NUMBER_OF_FIELDS
  Dakota, 137

objective
  Dakota::Minimizer, 532
objective_eval
  Dakota::NCSUOptimizer, 584
objective_gradient
  Dakota::Minimizer, 532
objective_hessian
  Dakota::Minimizer, 533
objective_reduction
  Dakota::Optimizer, 761
operator<<
Dakota::BoStream, 293
Dakota::CommandShell, 309
operator>>
  Dakota::BiStream, 290
operator=
  Dakota::Approximation, 269
  Dakota::Constraints, 331
  Dakota::Interface, 490
  Dakota::Iterator, 501
  Dakota::Model, 566
  Dakota::ProblemDescDB, 816
  Dakota::Strategy, 934
  Dakota::Variables, 995
optimizationType
  Dakota::CONMINOptimizer, 317
Optimizer
  Dakota::Optimizer, 759
OptionalValue
  Dakota::GetLongOpt, 464
OptType
  Dakota::GetLongOpt, 464
output_helper
  Dakota::CommandLineHandler, 307
output_version
  Dakota::CommandLineHandler, 306
pack_parameters_buffer
  Dakota::ConcurrentStrategy, 311
  Dakota::SequentialHybridStrategy, 889
  Dakota::Strategy, 934
pack_results_buffer
  Dakota::ConcurrentStrategy, 312
  Dakota::SequentialHybridStrategy, 889
  Dakota::Strategy, 934
ParallelLibrary
  Dakota::ParallelLibrary, 781, 782
ParamResponsePair
  Dakota::ParamResponsePair, 788
parse
  Dakota::GetLongOpt, 465
parse_inputs
  Dakota::ProblemDescDB, 816
peer_dynamic_schedule_evaluations
  Dakota::ApplicationInterface, 258
peer_dynamic_schedule_evaluations_nowait
  Dakota::ApplicationInterface, 259
peer_static_schedule_evaluations
  Dakota::ApplicationInterface, 257
penalty_merit
  Dakota::SurrBasedMinimizer, 958
perform_analysis
  Dakota, 133
perform_evaluation_impl
  Dakota::COLINApplication, 296
PerformIterations
  Dakota::JEGAOptimizer::Driver, 425
PMA2_constraint_eval
  Dakota::NonDLocalReliability, 709
PMA_constraint_eval
  Dakota::NonDLocalReliability, 709
PMA_objective_eval
  Dakota::NonDLocalReliability, 708
pop
  Dakota::Approximation, 270
  Dakota::PecosApproximation, 805
pop_approximation
  Dakota::ApproximationInterface, 278
post_process
  Dakota::ProblemDescDB, 816
post_run
  Dakota::COLINOptimizer, 301
  Dakota::DDACEDesignCompExp, 406
  Dakota::FSUDesignCompExp, 454
  Dakota::Iterator, 502
  Dakota::LeastSq, 517
  Dakota::Optimizer, 760
  Dakota::ParamStudy, 794
  Dakota::PSUADEDesignCompExp, 833
  Dakota::SNLLLeastSq, 913
pre_output
  Dakota::Analyzer, 242
pre_run
  Dakota::DDACEDesignCompExp, 406
  Dakota::FSUDesignCompExp, 454
  Dakota::Iterator, 501
  Dakota::ParamStudy, 794
  Dakota::PSUADEDesignCompExp, 833
primary_resp_differencer
  Dakota::Minimizer, 533
primary_resp_reducer
  Dakota::Optimizer, 761
primary_resp_weighter
  Dakota::LeastSq, 519
print_distribution_map
  Dakota::NonD, 627
print_distribution_mappings
  Dakota::NonD, 626
print_restart
  Dakota, 135
print_restart_tabular
  Dakota, 135
print_results
  Dakota::Analyzer, 242
  Dakota::Iterator, 503
  Dakota::LeastSq, 518
  Dakota::NonDInterval, 687
  Dakota::Optimizer, 760
  Dakota::PStudyDACE, 829
  Dakota::RichExtrapVerification, 881
  Dakota::SurrBasedMinimizer, 956
  Dakota::Verification, 998
print_sobol_indices
  Dakota::Analyzer, 244
printControl
  Dakota::CONMINOptimizer, 317
  Dakota::DOTOptimizer, 422
prior_density
  Dakota::NonDDREAMBayesCalibration, 649
prior_sample
  Dakota::NonDDREAMBayesCalibration, 649
probability
  Dakota::NonDLocalReliability, 712
problem_size
  Dakota::NonDDREAMBayesCalibration, 649
ProblemDescDB
  Dakota::ProblemDescDB, 815
PRPMultiIndexCache
  Dakota, 131
PRPMultiIndexQueue
  Dakota, 131
PSUADEDesignCompExp
  Dakota::PSUADEDesignCompExp, 833
python_convert_int
  Dakota::PythonInterface, 836
quantify_uncertainty
  Dakota::NonDDREAMBayesCalibration, 649
  Dakota::NonDGPlmpSampling, 673
  Dakota::NonDGPMsABayesCalibration, 676
  Dakota::NonDIncremLHSSampling, 679
  Dakota::NonDLHSSampling, 694
  Dakota::NonDQUESOBayesCalibration, 731
randomSeed
  Dakota::NonDGPMsABayesCalibration, 677
rawResponseMap
  Dakota::Interface, 491
read
  Dakota::ParamResponsePair, 788
  Dakota::ResponseRep, 863–865
read_annotated
  Dakota::ResponseRep, 863
read_neutral
  Dakota, 135
read_tabular
  Dakota::MixedVariables, 539
  Dakota::RelaxedVariables, 853
  Dakota::ResponseRep, 864
read_variables_responses
  Dakota::Analyzer, 243
realCntlParmArray
  Dakota::DOTOptimizer, 422
rebuild
  Dakota::Approximation, 270
  Dakota::PecosApproximation, 804
rebuild_approximation
  Dakota::ApproximationInterface, 278
RecastModel
  Dakota::RecastModel, 846
RecordResponses
  Dakota::JEGAOptimizer::Evaluator, 441
recv
  Dakota::APPSEvalMgr, 282
reduce_model
  Dakota::Optimizer, 761
reduced_space_uq
  Dakota::EfficientSubspaceMethod, 432
RelaxedVarConstraints
  Dakota::RelaxedVarConstraints, 851
RelaxedVariables
  Dakota::RelaxedVariables, 853
repair_restart
  Dakota, 135
requestVector
  Dakota::ActiveSet, 237
reset
  Dakota::ResponseRep, 866
reset_inactive
  Dakota::ResponseRep, 866
reshape
  Dakota::Constraints, 331
  Dakota::MixedVarConstraints, 537
  Dakota::RelaxedVarConstraints, 851
  Dakota::ResponseRep, 866
resize_best_resp_array
  Dakota::Minimizer, 533
resize_best_vars_array
  Dakota::Minimizer, 533
INDEX

resolve_inputs
  Dakota::ParallelLibrary, 784
resolve_samples_symbols
  Dakota::DDACEDesignCompExp, 406
Response
  Dakota::Response, 858
response_mapping
  Dakota::Interface, 490
  Dakota::NestedModel, 593
response_modify_n2s
  Dakota::Minimizer, 535
response_modify_s2n
  Dakota::Minimizer, 531
responseMode
  Dakota::SurrogateModel, 963
ResponseRep
  Dakota::ResponseRep, 862, 863
restart_util.cpp, 1011
  main, 1011
restore
  Dakota::Approximation, 270
  Dakota::PecosApproximation, 805
restore_approximation
  Dakota::ApproximationInterface, 278
restore_available
  Dakota::ApproximationInterface, 278
retrieve
  Dakota::GetLongOpt, 465
returns_multiple_points
  Dakota::COLINOptimizer, 301
  Dakota::JEGAOptimizer, 513
RIA_constraint_eval
  Dakota::NonDLocalReliability, 708
RIA_objective_eval
  Dakota::NonDLocalReliability, 708
run
  Dakota::Iterator, 502
  Dakota::LeastSq, 517
  Dakota::NonD, 625
  Dakota::Optimizer, 760
  Dakota::PStudyDACE, 829
  Dakota::SurrBasedMinimizer, 956
  Dakota::Verification, 998
run_dakota_data
  Dakota, 133
run_dakota_mixed
  library_mode.cpp, 1007
run_dakota_parse
  library_mode.cpp, 1007
run_iterator
  Dakota::Iterator, 503
  Dakota::Strategy, 935
run_sequential
  Dakota::SequentialHybridStrategy, 890
run_sequential_adaptive
  Dakota::SequentialHybridStrategy, 890
S
  Dakota::CONMINOptimizer, 319
sample_likelihood
  Dakota::NonDDREAMBayesCalibration, 649
sampling_reset
  Dakota::NonDCubature, 645
  Dakota::NonDQuadrature, 728
  Dakota::NonDSampling, 739
  Dakota::NonDSParseGrid, 745
SCAL
  Dakota::CONMINOptimizer, 320
scale_model
  Dakota::Minimizer, 530
schedule_iterators
  Dakota::Strategy, 936
SCL_FIELD_NAMES
  Dakota, 233
SCL_NUMBER_OF_FIELDS
  Dakota, 233
secondary_resp_copier
  Dakota::Minimizer, 531
secondary_resp_scaler
  Dakota::Minimizer, 534
self_schedule_iterators
  Dakota::Strategy, 936
send_data_using_get
  Dakota::TrackerHTTP, 983
send_data_using_post
  Dakota::TrackerHTTP, 983
separable_combine
  Dakota::TestDriverInterface, 979
SeparateVariables
  Dakota::JEGAOptimizer::Evaluator, 441
serve_analyses_asynch
  Dakota::ProcessHandleApplicInterface, 827
serve_analyses_synch
  Dakota::ApplicationInterface, 256
serve_evaluations
  Dakota::ApplicationInterface, 255
serve_evaluations_asynch
  Dakota::ApplicationInterface, 261
serve_evaluations_asynch_peer

Dakota Version 5.4 Developers Manual generated on November 7, 2013
Dakota::ApplicationInterface, 261
serve_evaluations_synch
Dakota::ApplicationInterface, 260
serve_evaluations_synch_peer
Dakota::ApplicationInterface, 260
serve_iterators
Dakota::Strategy, 936
set_apps_parameters
Dakota::APPSOptimizer, 285
set_communicators_checks
Dakota::ApplicationInterface, 256
Dakota::DirectApplicInterface, 412
Dakota::ProcessHandleApplicInterface, 825
Dakota::SysCallApplicInterface, 967
set_compare
Dakota, 134
set_problem
Dakota::COLINApplication, 295
set_rng
Dakota::COLINOptimizer, 301
set_solver_parameters
Dakota::COLINOptimizer, 301
set_u_to_x_mapping
Dakota::NonD, 627
SharedVariablesDataRep
Dakota::SharedVariablesDataRep, 902
shubert1D
Dakota::TestDriverInterface, 979
SIM, 234
SIM::ParallelDirectApplicInterface, 765
test_local_evaluations, 766
SIM::SerialDirectApplicInterface, 892
test_local_evaluations, 893
smap
Dakota, 136
smooth_herbie
Dakota::TestDriverInterface, 979
smooth_herbie1D
Dakota::TestDriverInterface, 978
SNLLOptimizer
Dakota::SNLLOptimizer, 920
solver_setup
Dakota::COLINOptimizer, 301
spawn_analysis_to_shell
Dakota::SysCallApplicInterface, 968
spawn_evaluation_impl
Dakota::COLINApplication, 295
spawn_evaluation_to_shell
Dakota::SysCallApplicInterface, 967
spawn_input_filter_to_shell
Dakota::SysCallApplicInterface, 968
spawn_output_filter_to_shell
Dakota::SysCallApplicInterface, 968
specify_outputs_restart
Dakota::ParallelLibrary, 782
split_filenames
Dakota::ParallelLibrary, 784
start_dakota_heartbeat
Dakota, 131
start_grid_computing
Dakota, 133
stop_evaluation_servers
Dakota::ApplicationInterface, 255
stop_grid_computing
Dakota, 133
Strategy
Dakota::Strategy, 933
submit
Dakota::APPSEvalMgr, 282
subModel
Dakota::NestedModel, 594
subordinate_iterator
Dakota::Model, 566
subordinate_model
Dakota::Model, 566
subordinate_models
Dakota::Model, 569
SurfpackApproximation
Dakota::SurfpackApproximation, 941
surrogate_model
Dakota::Model, 567
surrogates_to_surf_data
Dakota::SurfpackApproximation, 941
synch
Dakota::ApplicationInterface, 254
synch_nowait
Dakota::ApplicationInterface, 255
synchronize_derivatives
Dakota::Model, 573
synchronous_local_analyses
Dakota::ProcessApplicInterface, 821
synchronous_local_analyses
Dakota::DirectApplicInterface, 412
Dakota::GridApplicInterface, 472
Dakota::ProcessHandleApplicInterface, 825
Dakota::SysCallApplicInterface, 967
synchronous_local_evaluations
Dakota::ApplicationInterface, 259
test_local_evaluations
   Dakota::SysCallApplicInterface, 966
SIM::ParallelDirectApplicInterface, 766
SIM::SerialDirectApplicInterface, 893

ToDoubleMatrix
   Dakota::JEGAOptimizer, 512

tr_ratio_check
   Dakota::SurrBasedLocalMinimizer, 951

truth_model
   Dakota::Model, 567

uncertain_vars_to_subspace
   Dakota::EfficientSubspaceMethod, 432

unpack_parameters_buffer
   Dakota::ConcurrentStrategy, 312
   Dakota::SequentialHybridStrategy, 889
   Dakota::Strategy, 934

unpack_results_buffer
   Dakota::ConcurrentStrategy, 312
   Dakota::SequentialHybridStrategy, 890
   Dakota::Strategy, 934

update
   Dakota::ResponseRep, 865
   Dakota::DataFitSurrModel, 345
   Dakota::ApproximationInterface, 276
   Dakota::DataFitSurrModel, 342

update_augmented_lagrange_multipliers
   Dakota::SurrBasedMinimizer, 957

update_filter
   Dakota::SurrBasedMinimizer, 957

update_from_actual_model
   Dakota::DataFitSurrModel, 346

update_from_sub_model
   Dakota::RecastModel, 848

update_from_subordinate_model
   Dakota::Model, 567

update_lagrange_multipliers
   Dakota::SurrBasedMinimizer, 957

update_level_data
   Dakota::NonDLocalReliability, 711

update_mpp_search_data
   Dakota::NonDLocalReliability, 711

update_partial
   Dakota::ResponseRep, 865

update_penalty
   Dakota::SurrBasedLocalMinimizer, 951

update_quasi_hessians
   Dakota::Model, 574

update_response
   Dakota::Model, 573

usage
   Dakota::GetLongOpt, 466

useDerivs
   Dakota::NonDExpansion, 658

Valueless
   Dakota::GetLongOpt, 464

var_mp_ebound
   Dakota, 233

var_mp_check_cau
   Dakota, 231

var_mp_check_ceu
   Dakota, 232

var_mp_check_cv
   Dakota, 231

var_mp_check_dau
   Dakota, 232

var_mp_check_deau
   Dakota, 232

var_mp_check_dei
   Dakota, 232

var_mp_check_deur
   Dakota, 232

var_mp_check_dset
   Dakota, 231

var_mp_drange
   Dakota, 233

Variables
   Dakota::Variables, 994, 995

variables_scaler
   Dakota::Minimizer, 534

variance_based_decomp
   Dakota::Analyzer, 243

vars_u_to_x_mapping
   Dakota::NonD, 626

vars_x_to_u_mapping
   Dakota::NonD, 627

view_aleatory_uncertain_counts
   Dakota::NonDSampling, 740

view_design_counts
   Dakota::NonDSampling, 740

view_epistemic_uncertain_counts
   Dakota::NonDSampling, 740

view_uncertain_counts
   Dakota::NonDSampling, 741

Vlch
Dakota, 230
VLI
  Dakota, 231
VLR
  Dakota, 230
volumetric_quality
  Dakota::PStudyDACE, 830
wait_local_evaluations
  Dakota::GridApplicInterface, 472
  Dakota::SysCallApplicInterface, 966
weight_model
  Dakota::LeastSq, 518
write
  Dakota::ParamResponsePair, 788
  Dakota::ResponseRep, 863–865
write.annotated
  Dakota::ResponseRep, 864
write.tabular
  Dakota::ResponseRep, 864