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

Version 4.1 Developers Manual

Michael S. Eldred, Brian M. Adams, David M. Gay, Laura P. Swiler
Optimization and Uncertainty Estimation Department

Karen Haskell
Scientific Applications and User Support Department

William J. Bohnhoff
Radiation Transport Department

John P. Eddy
System Sustainment and Readiness Technologies Department

William E. Hart, Jean-Paul Watson
Discrete Algorithms and Math Department

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

Josh D. Griffin, Patty D. Hough, Tammy G. Kolda, Pamela J. Williams
Mathematics, Informatics and Decision Sciences Department

Monica L. Martinez-Canales
Advanced Software Research and Development 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 finite element 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 actual source code and provides detailed class documentation, including all member functions and attributes.
## Contents

1 DAKOTA Developers Manual 7
   1.1 Introduction ............................................. 7
   1.2 Overview of DAKOTA ........................................ 7
   1.3 Services .................................................. 11
   1.4 Additional Resources ...................................... 12

2 DAKOTA Namespace Index 15
   2.1 DAKOTA Namespace List ................................... 15

3 DAKOTA Hierarchical Index 17
   3.1 DAKOTA Class Hierarchy .................................. 17

4 DAKOTA Class Index 21
   4.1 DAKOTA Class List ........................................ 21

5 DAKOTA File Index 25
   5.1 DAKOTA File List .......................................... 25

6 DAKOTA Page Index 27
   6.1 DAKOTA Related Pages .................................... 27

7 DAKOTA Namespace Documentation 29
   7.1 Dakota Namespace Reference .............................. 29
   7.2 SIM Namespace Reference ................................ 61

8 DAKOTA Class Documentation 63
   8.1 ActiveSet Class Reference ................................. 63
   8.2 AllConstraints Class Reference ......................... 66
| 8.3  | AllVariables Class Reference                   | 70  |
| 8.4  | AnalysisCode Class Reference                   | 74  |
| 8.5  | Analyzer Class Reference                       | 78  |
| 8.6  | ApplicationInterface Class Reference           | 82  |
| 8.7  | Approximation Class Reference                  | 93  |
| 8.8  | ApproximationInterface Class Reference         | 100 |
| 8.9  | Array Class Template Reference                 | 104 |
| 8.10 | BaseConstructor Struct Reference               | 108 |
| 8.11 | BaseVector Class Template Reference            | 109 |
| 8.12 | BiStream Class Reference                       | 113 |
| 8.13 | BoStream Class Reference                       | 117 |
| 8.14 | COLINApplication Class Reference              | 120 |
| 8.15 | COLINOptimizer Class Template Reference        | 123 |
| 8.16 | ColinPoint Class Reference                     | 127 |
| 8.17 | CommandLineHandler Class Reference            | 128 |
| 8.18 | CommandShell Class Reference                   | 130 |
| 8.19 | ConcurrentStrategy Class Reference             | 132 |
| 8.20 | CONMINOptimizer Class Reference                | 135 |
| 8.21 | Constraints Class Reference                    | 143 |
| 8.22 | CtelRegexp Class Reference                     | 152 |
| 8.23 | DataFitSurrModel Class Reference               | 154 |
| 8.24 | DataInterface Class Reference                  | 162 |
| 8.25 | DataMethod Class Reference                     | 166 |
| 8.26 | DataModel Class Reference                      | 177 |
| 8.27 | DataResponses Class Reference                  | 180 |
| 8.28 | DataStrategy Class Reference                   | 184 |
| 8.29 | DataVariables Class Reference                  | 188 |
| 8.30 | DDACEDesignCompExp Class Reference             | 195 |
| 8.31 | DirectFnApplication Interface Class Reference  | 199 |
| 8.32 | DirectFnApplication Interface Class Reference  | 201 |
| 8.33 | DistinctConstraints Class Reference            | 207 |
| 8.34 | DistinctVariables Class Reference              | 211 |
| 8.35 | DOTOptimizer Class Reference                   | 216 |
| 8.36 | EffGlobalOptimizer Class Reference             | 221 |
CONTENTS

8.37 ErrorTable Struct Reference .................................................. 226
8.38 ForkAnalysisCode Class Reference ........................................... 227
8.39 ForkApplicInterface Class Reference ...................................... 229
8.40 FSUDesignCompExp Class Reference ........................................ 232
8.41 FunctionCompare Class Template Reference .............................. 236
8.42 GaussProcApproximation Class Reference .................................. 237
8.43 GenLaguerreOrthogPolynomial Class Reference ......................... 243
8.44 GetLongOpt Class Reference .................................................. 245
8.45 Graphics Class Reference ..................................................... 249
8.46 GridApplicInterface Class Reference ....................................... 253
8.47 HermiteOrthogPolynomial Class Reference ............................... 256
8.48 HierarchSurrModel Class Reference ......................................... 258
8.49 IDRProblemDescDB Class Reference ......................................... 262
8.50 Interface Class Reference ...................................................... 265
8.51 Iterator Class Reference ....................................................... 275
8.52 JacobiOrthogPolynomial Class Reference ................................. 286
8.53 JEGAOptimizer Class Reference ............................................. 288
8.54 JEGAOptimizer::Driver Class Reference ................................. 296
8.55 JEGAOptimizer::EvaluatorCreator Class Reference .................... 299
8.56 LaguerreOrthogPolynomial Class Reference .............................. 301
8.57 LeastSq Class Reference ....................................................... 303
8.58 LegendreOrthogPolynomial Class Reference .............................. 306
8.59 List Class Template Reference ................................................ 308
8.60 Matrix Class Template Reference ............................................ 312
8.61 MergedConstraints Class Reference ......................................... 315
8.62 MergedVariables Class Reference ........................................... 318
8.63 Minimizer Class Reference .................................................... 322
8.64 Model Class Reference ........................................................ 329
8.65 MPIPackBuffer Class Reference .............................................. 358
8.66 MPIUnpackBuffer Class Reference .......................................... 361
8.67 MultilevelOptStrategy Class Reference ..................................... 364
8.68 NCSUOptimizer Class Reference ............................................. 367
8.69 NestedModel Class Reference ................................................ 370
8.70 Nl2Misc Struct Reference ..................................................... 376
<table>
<thead>
<tr>
<th>Section</th>
<th>Class Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.71</td>
<td>NL2SOLLeastSq Class Reference</td>
<td>377</td>
</tr>
<tr>
<td>8.72</td>
<td>NLPQLPOptimizer Class Reference</td>
<td>380</td>
</tr>
<tr>
<td>8.73</td>
<td>NLSSOLLeastSq Class Reference</td>
<td>385</td>
</tr>
<tr>
<td>8.74</td>
<td>NoDBBaseConstructor Struct Reference</td>
<td>387</td>
</tr>
<tr>
<td>8.75</td>
<td>NonD Class Reference</td>
<td>388</td>
</tr>
<tr>
<td>8.76</td>
<td>NonDAadaptImpSampling Class Reference</td>
<td>400</td>
</tr>
<tr>
<td>8.77</td>
<td>NonDCubature Class Reference</td>
<td>404</td>
</tr>
<tr>
<td>8.78</td>
<td>NonDEvidence Class Reference</td>
<td>407</td>
</tr>
<tr>
<td>8.79</td>
<td>NonDGlobalReliability Class Reference</td>
<td>413</td>
</tr>
<tr>
<td>8.80</td>
<td>NonDIncremlHSSampling Class Reference</td>
<td>416</td>
</tr>
<tr>
<td>8.81</td>
<td>NonDIntegration Class Reference</td>
<td>419</td>
</tr>
<tr>
<td>8.82</td>
<td>NonDLHSSampling Class Reference</td>
<td>421</td>
</tr>
<tr>
<td>8.83</td>
<td>NonDLocalReliability Class Reference</td>
<td>424</td>
</tr>
<tr>
<td>8.84</td>
<td>NonDPolynomialChaos Class Reference</td>
<td>431</td>
</tr>
<tr>
<td>8.85</td>
<td>NonDQuadrature Class Reference</td>
<td>433</td>
</tr>
<tr>
<td>8.86</td>
<td>NonDReliability Class Reference</td>
<td>436</td>
</tr>
<tr>
<td>8.87</td>
<td>NonDSampling Class Reference</td>
<td>441</td>
</tr>
<tr>
<td>8.88</td>
<td>NPSOLOptimizer Class Reference</td>
<td>447</td>
</tr>
<tr>
<td>8.89</td>
<td>Optimizer Class Reference</td>
<td>450</td>
</tr>
<tr>
<td>8.90</td>
<td>OrthogonalPolynomial Class Reference</td>
<td>454</td>
</tr>
<tr>
<td>8.91</td>
<td>OrthogPolyApproximation Class Reference</td>
<td>459</td>
</tr>
<tr>
<td>8.92</td>
<td>ParallelConfiguration Class Reference</td>
<td>465</td>
</tr>
<tr>
<td>8.93</td>
<td>ParallelLevel Class Reference</td>
<td>467</td>
</tr>
<tr>
<td>8.94</td>
<td>ParallelLibrary Class Reference</td>
<td>470</td>
</tr>
<tr>
<td>8.95</td>
<td>ParamResponsePair Class Reference</td>
<td>481</td>
</tr>
<tr>
<td>8.96</td>
<td>ParamStudy Class Reference</td>
<td>485</td>
</tr>
<tr>
<td>8.97</td>
<td>ProblemDescDB Class Reference</td>
<td>488</td>
</tr>
<tr>
<td>8.98</td>
<td>PStudyDACE Class Reference</td>
<td>496</td>
</tr>
<tr>
<td>8.99</td>
<td>PSUADEDesignCompExp Class Reference</td>
<td>499</td>
</tr>
<tr>
<td>8.100</td>
<td>RecastBaseConstructor Struct Reference</td>
<td>502</td>
</tr>
<tr>
<td>8.101</td>
<td>RecastModel Class Reference</td>
<td>503</td>
</tr>
<tr>
<td>8.102</td>
<td>Response Class Reference</td>
<td>509</td>
</tr>
<tr>
<td>8.103</td>
<td>ResponseRep Class Reference</td>
<td>514</td>
</tr>
<tr>
<td>8.104</td>
<td>SingleMethodStrategy Class Reference</td>
<td>520</td>
</tr>
</tbody>
</table>
11.2 Rebuild IDR ................................................................. 616
11.3 Update IDRProblemDescDB.C in Dakota/src ....................... 616
11.4 Update ProblemDescDB.C in Dakota/src .............................. 618
11.5 Update Corresponding Data Classes ................................. 619
11.6 Use get_<data_type>() Functions ..................................... 620
11.7 Update the Documentation ............................................. 620

12 Interfacing with DAKOTA as a Library ................................. 623
  12.1 Introduction ............................................................. 623
  12.2 Problem database populated through input file parsing .......... 624
  12.3 Problem database populated through external means ............ 625
  12.4 Instantiating the strategy ............................................ 626
  12.5 Defining the direct application interface ........................ 626
  12.6 Executing the strategy ............................................... 628
  12.7 Retrieving data after a run ........................................ 628
  12.8 Summary ............................................................... 628

13 Performing Function Evaluations ....................................... 631
  13.1 Synchronous function evaluations ................................ 631
  13.2 Asynchronous function evaluations ................................ 631
  13.3 Analyses within each function evaluation ........................ 632
  13.4 Software Tools for DAKOTA Development ........................ 633
  13.5 Todo List ............................................................ 635
Chapter 1

DAKOTA Developers Manual

Author:

Michael S. Eldred, Brian M. Adams, Karen Haskell, William J. Bohnhoff, John P. Eddy, David M. Gay, Josh D. Griffin, William E. Hart, Patty D. Hough, Tamara G. Kolda, Monica L. Martinez-Canales, Laura P. Swiler, Jean-Paul Watson, Pamela J. Williams

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, and stochastic finite element methods, parameter estimation with nonlinear least squares methods, and sensitivity/variance analysis with design of experiments and parameter study capabilities. 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 problem-solving environment as well as a platform for rapid prototyping of new solution approaches.

The Developers Manual focuses on documentation of the class structures used by the DAKOTA system. It derives directly from annotation of the actual source code. For information on input command syntax, refer to the Reference Manual, and for a tour of DAKOTA features and capabilities, refer to the Users Manual.

1.2 Overview of DAKOTA

In the DAKOTA system, 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 small number 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 software components are presented in the following sections using a top-down order.

### 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.
- **MultilevelOptStrategy**: hybrid optimization using a succession of iterators employing a succession of models of varying fidelity. The best results obtained are passed from one iterator to the next.
- **SurrBasedOptStrategy**: surrogate-based optimization. Employs a single iterator with a SurrogateModel (either data fit or hierarchical). A sequence of approximate optimizations is performed, each of which involves build, optimize, and verify steps.
- **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**.

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

- Optimization: **Optimizer** provides a base class for the DOTOptimizer, CONMINOptimizer, NPSOLOptimizer, NLPQLPOptimizer, and SNLLOptimizer gradient-based optimization libraries and the
COLINOptimizer, JEGAOptimizer, NCSUOptimizer, and EffGlobalOptimizer nongradient-based optimization methods and libraries.

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

and the Analyzer classes include:

- Uncertainty quantification: NonD provides a base class for NonDReliability (reliability analysis), NonDEvidence (Dempster-Shafer Theory of Evidence), NonDPolynomialChaos (generalized polynomial chaos expansions), NonDSampling, and NonDIntegration. NonDReliability is further specialized with local and global methods (NonDLocalReliability and NonDGlobalReliability), NonDIntegration is further specialized with quadrature and cubature methods (NonDQuadrature and NonDCubature), and NonDSampling is further specialized with the NonDLHSSampling class for Latin hypercube and Monte Carlo sampling, the NonDIncremLHSSampling class for incremental Latin hypercube sampling, and NonDAdaptImpSampling for multimodal adaptive importance sampling.

- 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 DDACEDesignCompExp and FSUDesignCompExp, which provide for parameter space exploration through design and analysis of computer experiments. NonDLHSSampling from the uncertainty quantification branch also supports a design of experiments mode.

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, uncertain, and state variable types for continuous and discrete domain types. This hierarchy is specialized according to various views of the data.

- **DistinctVariables**: both variable and domain type distinctions are retained, i.e. separate arrays for design, uncertain, and state variables types and for continuous and discrete domains.

- **AllVariables**: variable types are combined and domain type distinction is retained, i.e. design, uncertain, and state variable types combined into a single continuous variables array and a single discrete variables array.

- **MergedVariables**: variable type distinction is retained and domain types are combined, i.e. continuous and discrete variables merged into continuous arrays (integrality is relaxed) for design, uncertain, and state variable types.

The variables view that is chosen depends on the type of iterative study. For design optimization and uncertainty quantification, for example, variable and domain type distinctions are important and a **DistinctVariables** view is used. For parameter studies and design of experiments, however, the variable type distinctions can be ignored and an **AllVariables** view is used.

The **Constraints** hierarchy manages bound, linear, and nonlinear constraints and utilizes the same specializations for managing bounds on the variables (see **DistinctConstraints**, **AllConstraints**, and **MergedConstraints**).

### 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

- **SysCallApplicInterface**: the simulation is invoked using a system call (the C function `system()`). Asynchronous invocation utilizes a background system call. Utilizes the **SysCallAnalysisCode** class to define syntax for input filter, analysis code, output filter, or combined spawning, which in turn utilize the **CommandShell** utility.

- **ForkApplicInterface**: the simulation is invoked using a fork (the `fork/exec/wait` family of functions). Asynchronous invocation utilizes a nonblocking fork. Utilizes the **ForkAnalysisCode** class for lower level fork operations.
1.3 Services

- **GridApplicInterface**: the simulation is invoked using distributed resource facilities. This capability is experimental and still under development. The design is evolving into the use of Condor and/or Globus tools.

and the following semi-intrusive approach

- **DirectFnApplicInterface**: 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).

are supported. 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 data fit approximation case, global, multipoint, or local 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 permits the mixing of approximation types (using the Approximation derived classes: SurfpackApproximation (provides kriging, neural network, MARS, polynomial regression, and radial basis functions), GaussProcApproximation, OrthogPolyApproximation (utilizes an array of OrthogonalPolynomial instances to manage multivariate orthogonal polynomials from the Wiener-Askey scheme), TANA3Approximation, and TaylorApproximation).

Note: in the data fit approximation case, DataFitSurrModel provides the bulk of the surrogate management logic. It contains an ApproximationInterface object which provides the approximate parameter to response mappings. In the hierarchical approximation case, an ApproximationInterface object is not used since HierarchSurrModel uses low and high fidelity models to manage surrogate construction/usage.

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 an objective function 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 ConcurrentStrategy, ApplicationInterface, and ForkApplicInterface.
• Parsing: DAKOTA employs the Input Deck Reader (IDR) parser to retrieve information from
user input files. Parsing options are processed in CommandLineHandler and parsing occurs in
ProblemDescDB::manage_inputs() called from main.C. IDR uses the keyword handlers in the
IDRProblemDescDB 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.C, 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 shar-
ing 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 hierar-
chies: Strategy, Iterator, Model, Variables, Constraints, Interface, ProblemDescDB, Approximation, and
OrthogonalPolynomial. The latter idiom provides for memory efficiency in data-intensive classes which do
not involve a class hierarchy. Currently, only the Response class uses 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: DAKOTA provides 2D iteration history graphics using Motif widgets and 3D surface plotting
graphics from the PLPLOT package. Graphics data can also be catalogued in a tabular data file for post-
processing with 3rd party tools such as Matlab, Tecplot, etc. All of these capabilities are encapsulated
within the Graphics class.

1.4 Additional Resources

Additional development resources include:

• Recommended Practices for DAKOTA Development

• Software Tools for DAKOTA Development

• Instructions for Modifying DAKOTA’s Input Specification

• In addition to its normal usage as a stand-alone application, DAKOTA may be interfaced as an algorithm
library as described in Interfacing with DAKOTA as a Library.

• The execution of function evaluations is a core component of DAKOTA involving several class hierarchies.
An overview of the classes and member functions involved in performing these evaluations is provided in
Performing Function Evaluations.
Chapter 2

DAKOTA Namespace Index

2.1 DAKOTA Namespace List

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

- Dakota (The primary namespace for DAKOTA) .................................................. 29
- SIM (Plug facilities into DAKOTA) ................................................................. 61
Chapter 3

DAKOTA Hierarchical Index

3.1 DAKOTA Class Hierarchy

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

- ActiveSet .......................................................... 63
- AnalysisCode ...................................................... 74
  - ForkAnalysisCode .............................................. 227
  - SysCallAnalysisCode ......................................... 575
- Approximation .................................................... 93
  - GaussProcApproximation ................................... 237
  - OrthogPolyApproximation .................................. 459
  - SurfpackApproximation ...................................... 550
  - TANA3Approximation ......................................... 580
  - TaylorApproximation ......................................... 583
- Array ................................................................. 104
- BaseConstructor ................................................... 108
- BaseVector .......................................................... 109
  - Matrix ............................................................ 312
  - Vector ............................................................ 596
- BaseVector< Dakota::BaseVector< T > > .......................... 109
- BiStream ............................................................ 113
- BoStream ............................................................ 117
- COLINApplication ............................................... 120
- ColinPoint .......................................................... 127
- CommandShell ..................................................... 130
- Constraints .......................................................... 143
  - AllConstraints .................................................. 66
  - DistinctConstraints .......................................... 207
  - MergedConstraints ............................................. 315
- CtelRegexp .......................................................... 152
- DataInterface ....................................................... 162
- DataMethod .......................................................... 166
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataModel</td>
<td>177</td>
</tr>
<tr>
<td>DataResponses</td>
<td>180</td>
</tr>
<tr>
<td>DataStrategy</td>
<td>184</td>
</tr>
<tr>
<td>DataVariables</td>
<td>188</td>
</tr>
<tr>
<td>ErrorTable</td>
<td>226</td>
</tr>
<tr>
<td>FunctionCompare</td>
<td>236</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>245</td>
</tr>
<tr>
<td>CommandLineHandler</td>
<td>128</td>
</tr>
<tr>
<td>Graphics</td>
<td>249</td>
</tr>
<tr>
<td>Interface</td>
<td>265</td>
</tr>
<tr>
<td>ApplicationInterface</td>
<td>82</td>
</tr>
<tr>
<td>DirectFnApplicInterface</td>
<td>201</td>
</tr>
<tr>
<td>DirectFnApplicInterface</td>
<td>199</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>229</td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>253</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
<td>577</td>
</tr>
<tr>
<td>ApproximationInterface</td>
<td>100</td>
</tr>
<tr>
<td>Iterator</td>
<td>275</td>
</tr>
<tr>
<td>Analyzer</td>
<td>78</td>
</tr>
<tr>
<td>NonD</td>
<td>388</td>
</tr>
<tr>
<td>NonDEvidence</td>
<td>407</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>419</td>
</tr>
<tr>
<td>NonDCubature</td>
<td>404</td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>433</td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>431</td>
</tr>
<tr>
<td>NonDReliability</td>
<td>436</td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
<td>413</td>
</tr>
<tr>
<td>NonDLocalReliability</td>
<td>424</td>
</tr>
<tr>
<td>NonDSampling</td>
<td>441</td>
</tr>
<tr>
<td>NonDAdeptImpSampling</td>
<td>400</td>
</tr>
<tr>
<td>NonDIncrementLHSampling</td>
<td>416</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>421</td>
</tr>
<tr>
<td>PStudyDACE</td>
<td>496</td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>195</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>232</td>
</tr>
<tr>
<td>ParamStudy</td>
<td>485</td>
</tr>
<tr>
<td>PSUADEDesignCompExp</td>
<td>499</td>
</tr>
<tr>
<td>Minimizer</td>
<td>322</td>
</tr>
<tr>
<td>LeastSq</td>
<td>303</td>
</tr>
<tr>
<td>NL2SOLLeastSq</td>
<td>377</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>385</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>528</td>
</tr>
<tr>
<td>Optimizer</td>
<td>450</td>
</tr>
<tr>
<td>COLINOptimizer</td>
<td>123</td>
</tr>
<tr>
<td>CONMINOptimizer</td>
<td>135</td>
</tr>
<tr>
<td>DOTOptimizer</td>
<td>216</td>
</tr>
<tr>
<td>EffGlobalOptimizer</td>
<td>221</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>288</td>
</tr>
<tr>
<td>Class Name</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>367</td>
</tr>
<tr>
<td>NLQPLPOptimizer</td>
<td>380</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>447</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>532</td>
</tr>
<tr>
<td>JEGAOptimizer::Driver</td>
<td>296</td>
</tr>
<tr>
<td>JEGAOptimizer::EvaluatorCreator</td>
<td>299</td>
</tr>
<tr>
<td>List</td>
<td>308</td>
</tr>
<tr>
<td>Model</td>
<td>329</td>
</tr>
<tr>
<td>NestedModel</td>
<td>370</td>
</tr>
<tr>
<td>RecastModel</td>
<td>503</td>
</tr>
<tr>
<td>SingleModel</td>
<td>522</td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>569</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
<td>154</td>
</tr>
<tr>
<td>HierarchSurrModel</td>
<td>258</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>358</td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>361</td>
</tr>
<tr>
<td>Nl2Misc</td>
<td>376</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>387</td>
</tr>
<tr>
<td>OrthogonalPolynomial</td>
<td>454</td>
</tr>
<tr>
<td>GenLaguerreOrthogPolynomial</td>
<td>243</td>
</tr>
<tr>
<td>HermiteOrthogPolynomial</td>
<td>256</td>
</tr>
<tr>
<td>JacobiOrthogPolynomial</td>
<td>286</td>
</tr>
<tr>
<td>LaguerreOrthogPolynomial</td>
<td>301</td>
</tr>
<tr>
<td>LegendreOrthogPolynomial</td>
<td>306</td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>465</td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>467</td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>470</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>481</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>488</td>
</tr>
<tr>
<td>IDRProblemDescDB</td>
<td>262</td>
</tr>
<tr>
<td>RecastBaseConstructor</td>
<td>502</td>
</tr>
<tr>
<td>Response</td>
<td>509</td>
</tr>
<tr>
<td>ResponseRep</td>
<td>514</td>
</tr>
<tr>
<td>SNLLBase</td>
<td>525</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>528</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>532</td>
</tr>
<tr>
<td>SOLBase</td>
<td>539</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>385</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>447</td>
</tr>
<tr>
<td>Strategy</td>
<td>542</td>
</tr>
<tr>
<td>ConcurrentStrategy</td>
<td>132</td>
</tr>
<tr>
<td>MultiLevelOptStrategy</td>
<td>364</td>
</tr>
<tr>
<td>SingleMethodStrategy</td>
<td>520</td>
</tr>
<tr>
<td>SurrBasedOptStrategy</td>
<td>554</td>
</tr>
<tr>
<td>String</td>
<td>547</td>
</tr>
<tr>
<td>SurrogateDataPoint</td>
<td>565</td>
</tr>
<tr>
<td>SurrogateDataPointRep</td>
<td>567</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Variables</td>
<td>585</td>
</tr>
<tr>
<td>AllVariables</td>
<td>70</td>
</tr>
<tr>
<td>DistinctVariables</td>
<td>211</td>
</tr>
<tr>
<td>MergedVariables</td>
<td>318</td>
</tr>
<tr>
<td>VariablesUtil</td>
<td>594</td>
</tr>
<tr>
<td>AllConstraints</td>
<td>66</td>
</tr>
<tr>
<td>AllVariables</td>
<td>70</td>
</tr>
<tr>
<td>DistinctConstraints</td>
<td>207</td>
</tr>
<tr>
<td>DistinctVariables</td>
<td>211</td>
</tr>
<tr>
<td>MergedConstraints</td>
<td>315</td>
</tr>
<tr>
<td>MergedVariables</td>
<td>318</td>
</tr>
</tbody>
</table>
## Chapter 4

### DAKOTA Class Index

#### 4.1 DAKOTA Class List

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

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveSet</td>
<td>(Active set request vector and the derivative variables vector)</td>
<td>63</td>
</tr>
<tr>
<td>AllConstraints</td>
<td>(Employs the all data view)</td>
<td>66</td>
</tr>
<tr>
<td>AllVariables</td>
<td>(All data view)</td>
<td>70</td>
</tr>
<tr>
<td>AnalysisCode</td>
<td>(Processes for managing simulations)</td>
<td>74</td>
</tr>
<tr>
<td>Analyzer</td>
<td>(Hierarchy)</td>
<td>78</td>
</tr>
<tr>
<td>ApplicationInterface</td>
<td>(Interfaces to simulation codes)</td>
<td>82</td>
</tr>
<tr>
<td>Approximation</td>
<td>(Base class for the approximation class hierarchy)</td>
<td>93</td>
</tr>
<tr>
<td>ApproximationInterface</td>
<td>(Approximations to simulation-based results)</td>
<td>100</td>
</tr>
<tr>
<td>Array</td>
<td>(Template class for the Dakota bookkeeping array)</td>
<td>104</td>
</tr>
<tr>
<td>BaseConstructor</td>
<td>(Dummy struct for overloading letter-envelope constructors)</td>
<td>108</td>
</tr>
<tr>
<td>BaseVector</td>
<td>(Base class for the Dakota::Matrix and Dakota::Vector classes)</td>
<td>109</td>
</tr>
<tr>
<td>BiStream</td>
<td>(Data types)</td>
<td>113</td>
</tr>
<tr>
<td>BoStream</td>
<td>(Data types)</td>
<td>117</td>
</tr>
<tr>
<td>COLINApplication</td>
<td></td>
<td>120</td>
</tr>
<tr>
<td>COLINOptimizer</td>
<td>(Wrapper class for optimizers defined using COLIN)</td>
<td>123</td>
</tr>
<tr>
<td>ColinPoint</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>CommandLineHandler</td>
<td>(Utility class for managing command line inputs to DAKOTA)</td>
<td>128</td>
</tr>
<tr>
<td>CommandShell</td>
<td>(Processes with system calls)</td>
<td>130</td>
</tr>
<tr>
<td>ConcurrentStrategy</td>
<td>(Strategy for multi-start iteration or pareto set optimization)</td>
<td>132</td>
</tr>
<tr>
<td>CONMINOptimizer</td>
<td>(Wrapper class for the CONMIN optimization library)</td>
<td>135</td>
</tr>
<tr>
<td>Constraints</td>
<td>(Base class for the variable constraints class hierarchy)</td>
<td>143</td>
</tr>
<tr>
<td>CtelRegexp</td>
<td></td>
<td>152</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
<td>(Data fit surrogates (global and local))</td>
<td>154</td>
</tr>
<tr>
<td>DataInterface</td>
<td>(Container class for interface specification data)</td>
<td>162</td>
</tr>
<tr>
<td>DataMethod</td>
<td>(Container class for method specification data)</td>
<td>166</td>
</tr>
<tr>
<td>DataModel</td>
<td>(Container class for model specification data)</td>
<td>177</td>
</tr>
<tr>
<td>DataResponses</td>
<td>(Container class for responses specification data)</td>
<td>180</td>
</tr>
<tr>
<td>DataStrategy</td>
<td>(Container class for strategy specification data)</td>
<td>184</td>
</tr>
<tr>
<td>DataVariables</td>
<td>(Container class for variables specification data)</td>
<td>188</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>(Wrapper class for the DDACE design of experiments library)</td>
<td></td>
</tr>
<tr>
<td>DirectFnApplicInterface</td>
<td>(Sample derived interface class for testing plug-ins using assign_rep())</td>
<td></td>
</tr>
<tr>
<td>DistinctConstraints</td>
<td>(Default data view (no variable or domain type array merging))</td>
<td></td>
</tr>
<tr>
<td>DistinctVariables</td>
<td>(Default data view (no variable or domain type array merging))</td>
<td></td>
</tr>
<tr>
<td>DOTOptimizer</td>
<td>(Wrapper class for the DOT optimization library)</td>
<td></td>
</tr>
<tr>
<td>EffGlobalOptimizer</td>
<td>(Implementation of Efficient Global Optimization algorithm)</td>
<td></td>
</tr>
<tr>
<td>ErrorTable</td>
<td>(Data structure to hold errors)</td>
<td></td>
</tr>
<tr>
<td>ForkAnalysisCode</td>
<td>(Simulations using forks)</td>
<td></td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>(Using forks)</td>
<td></td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>(Wrapper class for the FSUDace QMC/CVT library)</td>
<td></td>
</tr>
<tr>
<td>GaussProcApproximation</td>
<td>(Derived approximation class for Gaussian Process implementation)</td>
<td></td>
</tr>
<tr>
<td>GenLaguerreOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for generalized Laguerre polynomials)</td>
<td></td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>((Advanced Computer Research Institute, Lyon, France)</td>
<td></td>
</tr>
<tr>
<td>Graphics</td>
<td>(For post-processing with Matlab, Tecplot, etc)</td>
<td></td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>(Using grid services such as Condor or Globus)</td>
<td></td>
</tr>
<tr>
<td>HermiteOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Hermite polynomials)</td>
<td></td>
</tr>
<tr>
<td>HierarchSurrogateModel</td>
<td>(Hierarchical surrogates (models of varying fidelity))</td>
<td></td>
</tr>
<tr>
<td>IDRProblemDescDB</td>
<td>(The derived input file database utilizing the IDR parser)</td>
<td></td>
</tr>
<tr>
<td>Interface</td>
<td>(Base class for the interface class hierarchy)</td>
<td></td>
</tr>
<tr>
<td>Iterator</td>
<td>(Base class for the iterator class hierarchy)</td>
<td></td>
</tr>
<tr>
<td>JacobiOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Jacobi polynomials)</td>
<td></td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>(A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA))</td>
<td></td>
</tr>
<tr>
<td>JEGAOptimizer::Driver</td>
<td>(A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm)</td>
<td></td>
</tr>
<tr>
<td>JEGAOptimizer::EvaluatorCreator</td>
<td>(A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator)</td>
<td></td>
</tr>
<tr>
<td>LaguerreOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Laguerre polynomials)</td>
<td></td>
</tr>
<tr>
<td>LeastSq</td>
<td>(Base class for the nonlinear least squares branch of the iterator hierarchy)</td>
<td></td>
</tr>
<tr>
<td>LegendreOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Legendre polynomials)</td>
<td></td>
</tr>
<tr>
<td>List</td>
<td>(Template class for the Dakota bookkeeping list)</td>
<td></td>
</tr>
<tr>
<td>Matrix</td>
<td>(Template class for the Dakota numerical matrix)</td>
<td></td>
</tr>
<tr>
<td>MergedConstraints</td>
<td>(Merged data view)</td>
<td></td>
</tr>
<tr>
<td>MergedVariables</td>
<td>(Merged data view)</td>
<td></td>
</tr>
<tr>
<td>Minimizer</td>
<td>(Iterator hierarchy)</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>(Base class for the model class hierarchy)</td>
<td></td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>(Class for packing MPI message buffers)</td>
<td></td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>(Class for unpacking MPI message buffers)</td>
<td></td>
</tr>
<tr>
<td>MultiLevelOptStrategy</td>
<td>(Multiple models of varying fidelity)</td>
<td></td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>(Wrapper class for the NCSU DIRECT optimization library)</td>
<td></td>
</tr>
<tr>
<td>NestedModel</td>
<td>(Execution within every evaluation of the model)</td>
<td></td>
</tr>
<tr>
<td>NIM2Misc</td>
<td>(Auxiliary information passed to calcr and calcj via ur)</td>
<td></td>
</tr>
<tr>
<td>NL2SOLLeastSq</td>
<td>(Wrapper class for the NL2SOL nonlinear least squares library)</td>
<td></td>
</tr>
<tr>
<td>NLQPLPOptimizer</td>
<td>(Wrapper class for the NLQPLP optimization library, Version 2.0)</td>
<td></td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>(Wrapper class for the NLSSOL nonlinear least squares library)</td>
<td></td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>(Dummy struct for overloading constructors used in on-the-fly instantiations)</td>
<td></td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>NonD</td>
<td>Base class for all nondeterministic iterators (the DAKOTA/UQ branch)</td>
<td></td>
</tr>
<tr>
<td>NonDAadaptImpSampling</td>
<td>Class for the Adaptive Importance Sampling methods within DAKOTA</td>
<td></td>
</tr>
<tr>
<td>NonDCubature</td>
<td>Integrals over uncorrelated uniforms</td>
<td></td>
</tr>
<tr>
<td>NonDEvidence</td>
<td>Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ</td>
<td></td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
<td>Class for global reliability methods within DAKOTA/UQ</td>
<td></td>
</tr>
<tr>
<td>NonDIncrementLHSSampling</td>
<td>Performs incremental LHS sampling for uncertainty quantification</td>
<td></td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>Numerical integration points for evaluation of expectation integrals</td>
<td></td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>Performs LHS and Monte Carlo sampling for uncertainty quantification</td>
<td></td>
</tr>
<tr>
<td>NonDLocalReliability</td>
<td>Class for the reliability methods within DAKOTA/UQ</td>
<td></td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>Quantification</td>
<td></td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>Normals/uniforms/exponentials/betas/gammas</td>
<td></td>
</tr>
<tr>
<td>NonDReliability</td>
<td>Base class for the reliability methods within DAKOTA/UQ</td>
<td></td>
</tr>
<tr>
<td>NonDSampling</td>
<td>NonDIncrementLHSSampling, and NonDAadaptImpSampling</td>
<td></td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>Wrapper class for the NPSOL optimization library</td>
<td></td>
</tr>
<tr>
<td>Optimizer</td>
<td>Base class for the optimizer branch of the iterator hierarchy</td>
<td></td>
</tr>
<tr>
<td>OrthogonalPolynomial</td>
<td>Base class for the orthogonal polynomial class hierarchy</td>
<td></td>
</tr>
<tr>
<td>OrthogPolyApproximation</td>
<td>Approximation</td>
<td></td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>Collectively identify a particular multilevel parallel configuration</td>
<td></td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>Communicator partitioning</td>
<td></td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>Message passing within these levels</td>
<td></td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>(Evaluation id)</td>
<td></td>
</tr>
<tr>
<td>ParamStudy</td>
<td>Class for vector, list, centered, and multidimensional parameter studies</td>
<td></td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>(The database containing information parsed from the DAKOTA input file)</td>
<td></td>
</tr>
<tr>
<td>PStudyDACE</td>
<td>(Design of experiments methods)</td>
<td></td>
</tr>
<tr>
<td>PSUADEDesignCompExp</td>
<td>(Wrapper class for the PSUADE library)</td>
<td></td>
</tr>
<tr>
<td>RecastBaseConstructor</td>
<td>Instantiations</td>
<td></td>
</tr>
<tr>
<td>RecastModel</td>
<td>In order to recast the form of its inputs and/or outputs</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>(Response provides the handle class)</td>
<td></td>
</tr>
<tr>
<td>ResponseRep</td>
<td>(ResponseRep provides the body class)</td>
<td></td>
</tr>
<tr>
<td>SingleMethodStrategy</td>
<td>(Single model)</td>
<td></td>
</tr>
<tr>
<td>SingleModel</td>
<td>(Variables into responses)</td>
<td></td>
</tr>
<tr>
<td>SNLLLBase</td>
<td>(Base class for OPT++ optimization and least squares methods)</td>
<td></td>
</tr>
<tr>
<td>SNLLLLeastSq</td>
<td>(Wrapper class for the OPT++ optimization library)</td>
<td></td>
</tr>
<tr>
<td>SNLLLOptimizer</td>
<td>(Wrapper class for the OPT++ optimization library)</td>
<td></td>
</tr>
<tr>
<td>SOLBase</td>
<td>(Base class for Stanford SOL software)</td>
<td></td>
</tr>
<tr>
<td>Strategy</td>
<td>(Base class for the strategy class hierarchy)</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>(Dakota::String class, used as main string class for Dakota)</td>
<td></td>
</tr>
<tr>
<td>SurfpackApproximation</td>
<td>(Interface between Surfpack and Dakota)</td>
<td></td>
</tr>
<tr>
<td>SurrogateOptStrategy</td>
<td>(Strategy for provably-convergent surrogate-based optimization)</td>
<td></td>
</tr>
<tr>
<td>SurrogateDataPoint</td>
<td>(For defining a &quot;truth&quot; data point)</td>
<td></td>
</tr>
<tr>
<td>SurrogateDataPointRep</td>
<td>(Or body, may be shared by multiple SurrogateDataPoint handle instances)</td>
<td></td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>(Base class for surrogate models (DataFitSurrModel and HierarchSurrModel))</td>
<td></td>
</tr>
<tr>
<td>SysCallAnalysisCode</td>
<td>(Simulations using system calls)</td>
<td></td>
</tr>
<tr>
<td>SysCallApplicationInterface</td>
<td>(Using system calls)</td>
<td></td>
</tr>
<tr>
<td>TANA3Approximation</td>
<td>(Approximation (a multipoint approximation))</td>
<td></td>
</tr>
<tr>
<td>TaylorApproximation</td>
<td>(Series (a local approximation))</td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td>(Base class for the variables class hierarchy)</td>
<td></td>
</tr>
<tr>
<td>VariablesUtil</td>
<td>(Continuous and discrete variable domains)</td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td>(Template class for the Dakota numerical vector)</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

DAKOTA File Index

5.1 DAKOTA File List

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

- JEGAOptimizer.C (Contains the implementation of the JEGAOptimizer class) ........................................... 601
- JEGAOptimizer.H (Contains the definition of the JEGAOptimizer class) ............................................... 603
- keywordtable.C (File containing keywords for the strategy, method, model, variables, interface, and responses input specifications from dakota.input.spec) ........................................ 604
- main.C (File containing the main program for DAKOTA) ................................................................. 605
- restart_util.C (File containing the DAKOTA restart utility main program) ........................................... 606
Chapter 6

DAKOTA Page Index

6.1 DAKOTA Related Pages

Here is a list of all related documentation pages:

- Recommended Practices for DAKOTA Development .................................................. 609
- Instructions for Modifying DAKOTA's Input Specification ......................................... 615
- Interfacing with DAKOTA as a Library ........................................................................ 623
- Performing Function Evaluations .............................................................................. 631
- Software Tools for DAKOTA Development ................................................................. 633
- Todo List .................................................................................................................... 635
Chapter 7

DAKOTA Namespace Documentation

7.1 Dakota Namespace Reference

The primary namespace for DAKOTA.

Classes

- class AllConstraints
  
  employs the all data view.

- class AllVariables
  
  the all data view.

- class AnalysisCode
  
  processes for managing simulations.

- class ApplicationInterface
  
  interfaces to simulation codes.

- class ApproximationInterface
  
  approximations to simulation-based results.

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

- class GetLongOpt
  
  (Advanced Computer Research Institute, Lyon, France).
- **class** CommandLineHandler
  Utility class for managing command line inputs to DAKOTA.

- **class** CommandShell
  processes with system calls.

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

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

- **class** ActiveSet
  active set request vector and the derivative variables vector.

- **class** Analyzer
  hierarchy.

- **class** SurrogateDataPoint
  for defining a “truth” data point.

- **class** SurrogateDataPointRep
  or body, may be shared by multiple SurrogateDataPoint handle instances.

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

- **class** Array
  Template class for the Dakota bookkeeping array.

- **class** BaseVector
  Base class for the Dakota::Matrix and Dakota::Vector classes.

- **class** BiStream
  data types

- **class** BoStream
  data types

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

- **class** Graphics
  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 **List**  
  *Template class for the Dakota bookkeeping list.*

• class **FunctionCompare**

• class **Matrix**  
  *Template class for the Dakota numerical matrix.*

• class **Minimizer**  
  *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**  
  *design of experiments methods.*

• class **Response**  
  *Response provides the handle class.*

• class **ResponseRep**  
  *ResponseRep provides the body class.*

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

• class **String**  
  *Dakota::String class, used as main string class for Dakota.*

• class **Variables**  
  *Base class for the variables class hierarchy.*
- **class Vector**
  
  *Template class for the Dakota numerical vector.*

- **class DataFitSurrModel**
  
  *Data fit surrogates (global and local)*

- **class DataInterface**
  
  *Container class for interface specification data.*

- **class DataMethod**
  
  *Container class for method specification data.*

- **class DataModel**
  
  *Container class for model specification data.*

- **class DataResponses**
  
  *Container class for responses specification data.*

- **class DataStrategy**
  
  *Container class for strategy specification data.*

- **class DataVariables**
  
  *Container class for variables specification data.*

- **class DDACEDesignCompExp**
  
  *Wrapper class for the DDACE design of experiments library.*

- **class DirectFnApplicInterface**
  
  *and testers using direct procedure calls.*

- **class DistinctConstraints**
  
  *the default data view (no variable or domain type array merging).*

- **class DistinctVariables**
  
  *the default data view (no variable or domain type array merging).*

- **class DOTOptimizer**
  
  *Wrapper class for the DOT optimization library.*

- **class EffGlobalOptimizer**
  
  *Implementation of Efficient Global Optimization algorithm.*

- **class ForkAnalysisCode**
  
  *simulations using forks.*

- **class ForkApplicInterface**
using forks.

- class **FSUDesignCompExp**
  Wrapper class for the FSUDace QMC/CVT library.

- class **GaussProcApproximation**
  Derived approximation class for Gaussian Process implementation.

- class **GenLaguerreOrthogPolynomial**
  Derived orthogonal polynomial class for generalized Laguerre polynomials.

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

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

- struct **RecastBaseConstructor**
  instantiations.

- class **GridApplicInterface**
  using grid services such as Condor or Globus.

- class **HermiteOrthogPolynomial**
  Derived orthogonal polynomial class for Hermite polynomials.

- class **HierarchSurrModel**
  hierarchical surrogates (models of varying fidelity).

- class **IDRProblemDescDB**
  The derived input file database utilizing the IDR parser.

- class **JacobiOrthogPolynomial**
  Derived orthogonal polynomial class for Jacobi polynomials.

- class **JEGAOptimizer**
  A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

- class **LaguerreOrthogPolynomial**
  Derived orthogonal polynomial class for Laguerre polynomials.

- class **LegendreOrthogPolynomial**
  Derived orthogonal polynomial class for Legendre polynomials.

- class **MergedConstraints**
  the merged data view.
- class **MergedVariables**
  merged data view.

- class **MPIPackBuffer**
  Class for packing MPI message buffers.

- class **MPIUnpackBuffer**
  Class for unpacking MPI message buffers.

- class **MultilevelOptStrategy**
  multiple models of varying fidelity.

- class **NCSUOptimizer**
  Wrapper class for the NCSU DIRECT optimization library.

- class **NestedModel**
  execution within every evaluation of the model.

- struct **Nl2Misc**
  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 **NonDAadaptImpSampling**
  Class for the Adaptive Importance Sampling methods within DAKOTA.

- class **NonDCubature**
  integrals over uncorrelated uniforms.

- class **NonDEvidence**
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

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

- class **NonDIncremLHSSampling**
  Performs incremental LHS sampling for uncertainty quantification.
• class **NonDIntegration**
  
  *numerical integration points for evaluation of expectation integrals*

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

• class **NonDLocalReliability**
  
  *Class for the reliability methods within DAKOTA/UQ.*

• class **NonDPolynomialChaos**
  
  *quantification*

• class **NonDQuadrature**
  
  *normals/uniforms/exponentials/betas/gammas.*

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

• class **NonDSampling**
  
  *NonDIncremLHSSampling, and NonDAdaptImpSampling.*

• class **NPSOLOptimizer**
  
  *Wrapper class for the NPSOL optimization library.*

• class **OrthogonalPolynomial**
  
  *Base class for the orthogonal polynomial class hierarchy.*

• class **OrthogPolyApproximation**
  
  *approximation.*

• class **ParallelLevel**
  
  *communicator partitioning.*

• class **ParallelConfiguration**
  
  *collectively identify a particular multilevel parallel configuration.*

• class **ParallelLibrary**
  
  *message passing within these levels.*

• class **ParamResponsePair**
  
  *evaluation id.*

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

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

- class PSUADEDesignCompExp
  Wrapper class for the PSUADE library.

- class RecastModel
  in order to recast the form of its inputs and/or outputs.

- class SingleMethodStrategy
  single model.

- class SingleModel
  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 SurfpackApproximation
  Interface between Surfpack and Dakota.

- class SurrBasedOptStrategy
  Strategy for provably-convergent surrogate-based optimization.

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

- class SysCallAnalysisCode
  simulations using system calls.

- class SysCallApplicInterface
  using system calls.

- class TANA3Approximation
  approximation (a multipoint approximation).

- class TaylorApproximation
  series (a local approximation).
- class VariablesUtil
  
  *continuous and discrete variable domains.*

**Typedefs**

- typedef double Real
- typedef Vector<Real> RealVector
- typedef Vector<int> IntVector
- typedef BaseVector<Real> RealBaseVector
- typedef Matrix<Real> RealMatrix
- typedef Matrix<int> IntMatrix
- typedef std::deque<bool> BoolDeque
- typedef Array<BoolDeque> BoolDequeArray
- typedef Array<Real> RealArray
- typedef Array<int> IntArray
- typedef Array<short> ShortArray
- typedef Array<size_t> SizetArray
- typedef Array<SizetArray> Sizet2DArray
- typedef Array<String> StringArray
- typedef Array<Array<String>> String2DArray
- typedef Array<RealVector> RealVectorArray
- typedef Array<RealVectorArray> RealVector2DArray
- typedef Array<RealBaseVector> RealBaseVectorArray
- typedef Array<RealMatrix> RealMatrixArray
- typedef Array<Variables> VariablesArray
- typedef Array<Response> ResponseArray
- typedef Array<Model> ModelArray
- typedef Array<Iterator> IteratorArray
- typedef Array<ParamResponsePair> PRPArray
- typedef List<bool> BoolList
- typedef List<int> IntList
- typedef List<size_t> SizetList
- typedef List<Real> RealList
- typedef List<String> StringList
- typedef List<RealVector> RealVectorList
- typedef List<Variables> VariablesList
- typedef List<Interface> InterfaceList
- typedef List<Response> ResponseList
- typedef List<Model> ModelList
- typedef List<Iterator> IteratorList
- typedef List<ParamResponsePair> PRPList
- typedef std::set<int> IntSet
- typedef std::set<Real> RealSet
- typedef std::map<int, short> IntShortMap
typedef std::map<int, int> IntIntMap
typedef std::map<int, RealVector> IntRealVectorMap
typedef std::map<int, ActiveSet> IntActiveSetMap
typedef std::map<int, Variables> IntVariablesMap
typedef std::map<int, Response> IntResponseMap
typedef IntList::iterator ILIter
typedef IntList::const_iterator ILCIter
typedef IntSet::iterator ISIter
typedef IntSet::const_iterator ISCIter
typedef IntShortMap::iterator IntShMIter
typedef IntIntMap::iterator IntIntMIter
typedef IntIntMap::const_iterator IntIntMCIter
typedef IntRealVectorMap::iterator IntRVMIter
typedef IntActiveSetMap::iterator IntASMIter
typedef IntVariablesMap::iterator IntVarsMIter
typedef IntVariablesMap::const_iterator IntVarsMCIter
typedef IntResponseMap::iterator IntRespMIter
typedef IntResponseMap::const_iterator IntRespMCIter
typedef void(void, Optimizer1, char) dl_find_optimum_t
typedef void(void) dl_destructor_t
typedef double Real
typedef int(*)(char *analysis_driver_script, char *params_file, char *results_file) start_grid_computing_t
typedef int(*)(char *iteration_num) perform_analysis_t
typedef int get_jobs_completed_t()
typedef int(*)(char) stop_grid_computing_t()
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 void(*)(int *, int *, Real *, int *, Real *, int *, void *, Vf *) Calcrj
typedef void(*)(Vf)
Enumerations

- enum { OBJECTIVE, INEQUALITY_CONSTRAINT, EQUALITY_CONSTRAINT }
  
  define algebraic function types

- enum { SCALE_NONE = 0, SCALE_VALUE = 1, SCALE_LOG = 2 }

- enum { CDV, LINEAR, NONLIN, FN_LSQ }

- enum { DISALLOW, TARGET, BOUNDS }

- enum 
  
  DESIGN, NORMAL, LOGNORMAL, UNIFORM,
  LOGUNIFORM, TRIANGULAR, EXPONENTIAL, BETA,
  GAMMA, GUMBEL, FRECHET, WEIBULL,
  STATE }

- enum { NO_REFINE, IS, AIS, MMAIS }

- enum { PROBABILITIES, RELIABILITIES, GEN_RELIABILITIES }

- enum 
  
  ORIGINAL_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 LHSNames {
  
  LHS_NORMAL, LHS_LOGNORMAL, LHS_UNIFORM, LHS_LOGUNIFORM,
  LHS_WEIBULL, LHS_CONSTANT, LHS_USERDEFINED }

- enum {
  
  N_MEAN, N_STD_DEV, N_LWR_BND, N_UPR_BND,
  LN_MEAN, LN_STD_DEV, LN_ERR_FACT, LN_LWR_BND,
  LN_UPR_BND, U_LWR_BND, U_UPR_BND, LU_LWR_BND,
  LU_UPR_BND, T_MODE, T_LWR_BND, T_UPR_BND,
  E_BETA, B_ALPHA, B_BETA, B_LWR_BND,
  B_UPR_BND, GA_ALPHA, GA_BETA, GU_ALPHA,
  GU_BETA, F_ALPHA, F_BETA, W_ALPHA,
  W_BETA }

- enum { PENALTY, AUG_LAG, LAG }

- enum { EGRA_X, EGRA_U }

- enum {
  
  MV, AMV_X, AMV_U, AMV_PLUS_X,
  AMV_PLUS_U, TANA_X, TANA_U, NO_APPROX }

- enum { BREITUNG, HOHENRACK, HONG }

- enum {
  
  UNCERTAIN, UNCERTAIN_UNIFORM, ACTIVE, ACTIVE_UNIFORM,
  ALL, ALL_UNIFORM }
enum { IGNORE_RANKS, SET_RANKS, GET_RANKS, SET_GET_RANKS }

enum { HERMITE, LEGENDRE, LAGUERRE, JACOBI,
      GENERALIZED_LAGUERRE }

variable spec order of normal, uniform, exponential, beta, gamma)

enum { QUADRATURE, CUBATURE, POINT_COLLOCATION, SAMPLING }

solution approaches for calculating the polynomial chaos coefficients

enum EvalType { NLFEvaluator, CONEvaluator }

enumeration for the type of evaluator function

enum { EMPTY, MERGED_ALL, MIXED_ALL, MERGED_DISTINCT DESIGN,
       MERGED_DISTINCT UNCERTAIN, MERGED_DISTINCT STATE, MIXED_DISTINCT-
       DESIGN, MIXED_DISTINCT UNCERTAIN,
       MIXED_DISTINCT STATE }

Functions

bool operator==(const AllVariables &vars1, const AllVariables &vars2)

equality operator

CommandShell & flush (CommandShell &shell)

convenient shell manipulator function to "flush" the shell

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

equality operator

istream & operator>>(istream &s, ActiveSet &set)

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

ostream & operator<<(ostream &s, const ActiveSet &set)

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

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)

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

- template<class T> istream & operator>>(istream &s, Array<T> &data)
  
  global istream extraction operator for Vector

- template<class T> ostream & operator<<(ostream &s, const Array<T> &data)
  
  global ostream insertion operator for Array

- template<class T> BiStream & operator>>(BiStream &s, Array<T> &data)
  
  global BiStream extraction operator for Array

- template<class T> BoStream & operator<<(BoStream &s, const Array<T> &data)
  
  global BoStream insertion operator for Array

- template<class T> MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Array<T> &data)
  
  global MPIUnpackBuffer extraction operator for Array

- template<class T> MPIPackBuffer & operator<<(MPIPackBuffer &s, const Array<T> &data)
  
  global MPIPackBuffer insertion operator for Array

- istream & operator>>(istream &s, Constraints &con)
  
  istream extraction operator for Constraints

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

- template<class T> ostream & operator<<(ostream &s, const List<T> &data)
  
  global ostream insertion operator for List

- template<class T> MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, List<T> &data)
  
  global MPIUnpackBuffer extraction operator for List

- template<class T> MPIPackBuffer & operator<<(MPIPackBuffer &s, const List<T> &data)
  
  global MPIPackBuffer insertion operator for List

- template<class T> istream & operator>>(istream &s, Matrix<T> &data)
  
  global istream extraction operator for Matrix
• template<class T> ostream & operator<< (ostream &s, const Matrix<T> &data)
  global ostream insertion operator for Matrix

• template<class T> MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Matrix<T> &data)
  global MPIUnpackBuffer extraction operator for Matrix

• template<class T> MPIPackBuffer & operator<<(MPIPackBuffer &s, const Matrix<T> &data)
  global MPIPackBuffer insertion operator for Matrix

• 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

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

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

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

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

• MPIPackBuffer & operator<<(MPIPackBuffer &s, const String &data)
  Reads String from buffer.
• **MPIUnpackBuffer** & operator>>(MPIUnpackBuffer &s, String &data)
  
  Writes *String* to buffer.

• **String** operator+(const String &s1, const String &s2)
  
  Concatenate two Strings and return the resulting *String*.

• **String** operator+(const char *s1, const String &s2)
  
  Append a *String* to a char* and return the resulting *String*.

• **String** operator+(const String &s1, const char *s2)
  
  Append a char* to a *String* and return the resulting *String*.

• **String** operator+(const DAKOTA_BASE_STRING &s1, const String &s2)
  
  Append a *String* to a DAKOTA_BASE_STRING and return the resulting *String*.

• **String** operator+(const String &s1, const DAKOTA_BASE_STRING &s2)
  
  Append a DAKOTA_BASE_STRING to a *String* and return the resulting *String*.

• **String** toUpper(const String &str)
  
  Returns a *String* converted to upper case. Calls *String::upper*().

• **String** toLower(const String &str)
  
  Returns a *String* converted to lower case. Calls *String::lower*().

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

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

• istream & operator>>(istream &s, Variables &vars)
  
  istream extraction operator for *Variables*.

• ostream & operator<<(ostream &s, const Variables &vars)
  
  ostream insertion operator for *Variables*.

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

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

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

• **MPIPackBuffer** & operator<<(MPIPackBuffer &s, const Variables &vars)
**MPIPackBuffer** insertion operator for Variables.

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

- **template<class T>** istream & operator>>(istream &s, Vector<T> &data)
  global istream extraction operator for Vector

- **template<class T>** ostream & operator<<(ostream &s, const Vector<T> &data)
  global ostream insertion operator for Vector

- **template<class T> MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Vector<T> &data)
  global MPIUnpackBuffer extraction operator for Vector

- **template<class T> MPIPackBuffer & operator<<(MPIPackBuffer &s, const Vector<T> &data)
  global MPIPackBuffer insertion operator for Vector

- **bool operator==(** (const RealVector &drv1, const RealVector &drv2)
  equality operator for RealVector

- **bool operator==(** (const IntVector &div1, const IntVector &div2)
  equality operator for IntVector

- **bool operator==(** (const IntArray &dia1, const IntArray &dia2)
  equality operator for IntArray

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

- **bool operator==(** (const RealMatrix &drm1, const RealMatrix &drm2)
  equality operator for RealMatrix

- **bool operator==(** (const RealMatrixArray &drma1, const RealMatrixArray &drma2)
  equality operator for RealMatrixArray

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

- **void copy_data** (const NEWMAT::ColumnVector &cv, RealBaseVector &drbv)
  copy NEWMAT::ColumnVector to RealBaseVector

- **void copy_data** (const RealBaseVector &drbv, NEWMAT::ColumnVector &cv)
  copy RealBaseVector to NEWMAT::ColumnVector

- **void copy_data** (const RealArray &dra, NEWMAT::ColumnVector &cv)
  copy RealArray to NEWMAT::ColumnVector
- void copy_data (const RealMatrix &drm, NEWMAT::SymmetricMatrix &sm)
  \(\text{copy RealMatrix to NEWMAT::SymmetricMatrix}\)

- void copy_data (const RealMatrix &drm, NEWMAT::Matrix &m)
  \(\text{copy RealMatrix to NEWMAT::Matrix}\)

- void copy_data (const Epetra_SerialDenseVector &psdv, RealVector &drv)
  \(\text{copy Epetra_SerialDenseVector to RealVector}\)

- void copy_data (const Epetra_SerialDenseVector &psdv, RealBaseVector &drbv)
  \(\text{copy Epetra_SerialDenseVector to RealBaseVector}\)

- void copy_data (const Epetra_SerialDenseMatrix &psdm, RealMatrix &drm)
  \(\text{copy Epetra_SerialDenseMatrix to RealMatrix}\)

- void copy_data (const Epetra_SerialSymDenseMatrix &pssdm, RealMatrix &drm)
  \(\text{copy Epetra_SerialSymDenseMatrix to RealMatrix}\)

- void copy_data (const RealVector &drv, Epetra_SerialDenseVector &psdv)
  \(\text{copy RealVector to Epetra_SerialDenseVector}\)

- void copy_data (const RealArray &dra, Epetra_SerialDenseVector &psdv)
  \(\text{copy RealArray to Epetra_SerialDenseVector}\)

- void copy_data (const RealBaseVector &drbv, Epetra_SerialDenseVector &psdv)
  \(\text{copy RealBaseVector to Epetra_SerialDenseVector}\)

- void copy_data (const Real *, const int ptr_len, Epetra_SerialDenseVector &psdv)
  \(\text{copy Real\* to Epetra_SerialDenseVector}\)

- void copy_data (const RealMatrix &drm, Epetra_SerialDenseMatrix &psdm)
  \(\text{copy RealMatrix to Epetra_SerialDenseMatrix}\)

- void copy_data (const RealMatrix &drm, Epetra_SerialSymDenseMatrix &pssdm)
  \(\text{copy RealMatrix to Epetra_SerialSymDenseMatrix}\)

- void copy_data (const RealMatrixArray &drma, Array< Epetra_SerialSymDenseMatrix > &pssdma)
  \(\text{copy RealMatrixArray to Array< Epetra_SerialSymDenseMatrix >}\)

- void copy_data (const NEWMAT::ColumnVector &cv, Epetra_SerialDenseVector &psdv)
  \(\text{copy NEWMAT::ColumnVector to Epetra_SerialDenseVector}\)

- void copy_data (const DDaceSamplePoint &dsp, RealVector &drva)
  \(\text{copy DDACE point to RealVector}\)
- void `copy_data` (const std::vector<DDaceSamplePoint> &dspa, RealVectorArray &drva)
  
  copy DDACE point array to RealVectorArray
- void `copy_data` (const std::vector<DDaceSamplePoint> &dspa, Real *ptr, const int ptr_len)
  
  copy DDACE point array to Real*
- bool `operator!=` (const RealVector &drv1, const RealVector &drv2)
  
  inequality operator for RealVector
- bool `operator!=` (const IntVector &div1, const IntVector &div2)
  
  inequality operator for IntVector
- bool `operator!=` (const IntArray &dia1, const IntArray &dia2)
  
  inequality operator for IntArray
- bool `operator!=` (const ShortArray &dsa1, const ShortArray &dsa2)
  
  inequality operator for ShortArray
- bool `operator!=` (const RealMatrix &drm1, const RealMatrix &drm2)
  
  inequality operator for RealMatrix
- bool `operator!=` (const RealMatrixArray &drma1, const RealMatrixArray &drma2)
  
  inequality operator for RealMatrixArray
- bool `operator!=` (const StringArray &dsa1, const StringArray &dsa2)
  
  inequality operator for StringArray
- void `build_label` (String &label, const String &root_label, size_t tag)
  
  create a label by appending a numerical tag to the root_label
- void `build_labels` (StringArray &label_array, const String &root_label)
  
  label_array. Uses `build_label()`.
- void `build_labels_partial` (StringArray &label_array, const String &root_label, size_t start_index, size_t num_items)
  
  of entries in label_array. Uses `build_label()`.
- template<class T> ostream & operator<<(ostream &s, const std::set<T> &data)
  
  global ostream insertion operator for std::set
- template<class T> MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, std::set<T> &data)
  
  global MPIUnpackBuffer extraction operator for std::set
- template<class T> MPIPackBuffer & operator<<(MPIPackBuffer &s, const std::set<T> &data)
  
  global MPIPackBuffer insertion operator for std::set
template<class T> void copy_data (const T *ptr, const int ptr_len, Vector<T> &dv)
    copy T* to Vector<T>

template<class T> void copy_data (const T *ptr, const int ptr_len, BaseVector<T> &dbv)
    copy T* to BaseVector<T>

template<class T> void copy_data (const T *ptr, const int ptr_len, const String &ptr_type, Matrix<T> &dm, size_t nr, size_t nc)
    copy T* to Matrix<T>

template<class T> void copy_data (const T *ptr, const int ptr_len, const String &ptr_type, Array<Vector<T>> &dva, size_t num_vec, size_t vec_len)
    copy T* to Array<Vector<T> >

template<class T> void copy_data (const Vector<T> &dv, T *ptr, const int ptr_len)
    copy Vector<T> to T*

template<class T> void copy_data (const BaseVector<T> &dbv, T *ptr, const int ptr_len)
    copy BaseVector<T> to T*

template<class T> void copy_data (const Matrix<T> &dm, T *ptr, const int ptr_len, const String &ptr_type)
    copy Matrix<T> to T*

template<class T> void copy_data (const Array<Vector<T>> &dva, T *ptr, const int ptr_len, const String &ptr_type)
    copy Array<Vector<T> > to T*

template<class T> void copy_data (const Vector<T> &dv, Matrix<T> &dm, size_t nr, size_t nc)
    copy Vector<T> to Matrix<T>

template<class T> void copy_data (const Array<T> &da, Vector<T> &dv)
    copy Array<T> to Vector<T>
copy List<T> to Array<Array<T> >

- template<class T> void copy_data (const Array<Array<T> > &d2a, Array<T> &da)
  copy Array<Array<T> > to Array<T> (unroll 2D array into 1D array)

- template<class T> void copy_data_partial (const Vector<T> &dv1, size_t start_index, size_t num_items, Vector<T> &dv2)
  copy portion of first Vector<T> to all of second Vector<T>

- template<class T> void copy_data_partial (const Vector<T> &dv1, Vector<T> &dv2, size_t start_index)
  copy all of first Vector<T> to portion of second Vector<T>

- template<class T> void copy_data_partial (const Vector<T> &dv1, Vector<T> &dv2, size_t start_index1, size_t num_items, Vector<T> &dv2, size_t start_index2)
  copy portion of first Vector<T> to portion of second Vector<T>

- template<class T> void copy_data_partial (const Array<T> &da1, size_t start_index1, size_t num_items, Array<T> &da2)
  copy portion of first Array<T> to all of second Array<T>

- template<class T> void copy_data_partial (const Array<T> &da1, Array<T> &da2, size_t start_index1, size_t num_items, Array<T> &da2, size_t start_index2)
  copy portion of first Array<T> to portion of second Array<T>

- template<class T> void copy_data (const utilib::NumArray<T> &na, Vector<T> &dv)
  copy utilib::NumArray<T> to Vector<T>

- template<class T> void copy_data (const Vector<T> &dv, utilib::NumArray<T> &na)
  copy Vector<T> to utilib::NumArray<T>

- template<class T> void copy_data (const utilib::NumArray<T> &na, Array<T> &da)
  copy utilib::NumArray<T> to Array<T>

- template<class T> void copy_data (const List<T> &dl, utilib::NumArray<T> &na)
  copy List<T> to utilib::NumArray<T>

- template<class T> void copy_data (const TNT::Vector<T> &tntv, Vector<T> &dv)
  copy TNT::Vector<T> to Vector<T>

- template<class T> void copy_data (const Vector<T> &dv, TNT::Vector<T> &tntv)
  copy Vector<T> to TNT::Vector<T>
- template<class T> void `copy_data` (const T *ptr, const int ptr_len, TNT::Vector<T> &tntv)
  
  `copy T` to TNT::Vector<T>.

- template<class T> void `copy_data` (const Matrix<T> &dm, TNT::Matrix<T> &tntm)
  
  `copy Matrix<T>` to TNT::Matrix<T>.

- bool `data_interface_id_compare` (const DataInterface &di, const void *id)
  
  `global comparison function for DataInterface`

- MPIPackBuffer & operator<< (MPIPackBuffer &s, const DataInterface &data)
  
  `MPIPackBuffer insertion operator for DataInterface`.

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

- ostream & operator<< (ostream &s, const DataInterface &data)
  
  `ostream insertion operator for DataInterface`.

- bool `data_method_id_compare` (const DataMethod &dm, const void *id)
  
  `global comparison function for DataMethod`

- MPIPackBuffer & operator<< (MPIPackBuffer &s, const DataMethod &data)
  
  `MPIPackBuffer insertion operator for DataMethod`.

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

- ostream & operator<< (ostream &s, const DataMethod &data)
  
  `ostream insertion operator for DataMethod`.

- bool `data_model_id_compare` (const DataModel &dm, const void *id)
  
  `global comparison function for DataModel`

- MPIPackBuffer & operator<< (MPIPackBuffer &s, const DataModel &data)
  
  `MPIPackBuffer insertion operator for DataModel`.

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

- ostream & operator<< (ostream &s, const DataModel &data)
  
  `ostream insertion operator for DataModel`.

- bool `data_responses_id_compare` (const DataResponses &dr, const void *id)
  
  `global comparison function for DataResponses`

- MPIPackBuffer & operator<< (MPIPackBuffer &s, const DataResponses &data)
MPIPackBuffer insertion operator for DataResponses.

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

- ostream & operator<<(ostream &s, const DataResponses &data)
  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.

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

- bool data_variables_id_compare (const DataVariables &dv, const void *id)
  global comparison function for DataVariables

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataVariables &data)
  MPIPackBuffer insertion operator for DataVariables.

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

- ostream & operator<<(ostream &s, const DataVariables &data)
  ostream insertion operator for DataVariables

- bool operator==(const DistinctVariables &vars1, const DistinctVariables &vars2)
  equality operator

- int dlsolver_option (Opt_Info *)
- void abort_handler (int code)
  global function which handles serial or parallel aborts

- 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 *o)
RealMatrix const *linear_ineq_constraint_coeffs (Optimizer1 *o)
RealMatrix const *linear_eq_constraint_coeffs (Optimizer1 *o)
RealVector *bestFunctions (Optimizer1 *o)
void ComputeResponses (Optimizer1 *o, int mode, int n, double *x)
void GetFuncs (Optimizer1 *o, int m0, int m1, double *f)
void GetGrads (Optimizer1 *o, int m0, int m1, int n, int is, int js, double *g)
void GetContVars (Optimizer1 *o, int n, double *x)
void SetBestContVars (Optimizer1 *o, int n, double *x)
void *dl_constructor (Optimizer1 *, Dakota_funcs *, dl_find_optimum_t *, dlDestructor_t *)
static RealVector const *continuous_lower_bounds1 (Optimizer1 *o)
static RealVector const *continuous_upper_bounds1 (Optimizer1 *o)
static RealVector const *nonlinear_ineq_constraint_lower_bounds1 (Optimizer1 *o)
static RealVector const *nonlinear_ineq_constraint_upper_bounds1 (Optimizer1 *o)
static RealVector const *linear_ineq_constraint_lower_bounds1 (Optimizer1 *o)
static RealVector const *linear_ineq_constraint_upper_bounds1 (Optimizer1 *o)
static RealVector const *linear_eq_constraint_targets1 (Optimizer1 *o)
static RealMatrix const *linear_eq_constraint_coeffs1 (Optimizer1 *o)
static RealMatrix const *linear_ineq_constraint_coeffs1 (Optimizer1 *o)
static void ComputeResponses1 (Optimizer1 o, int mode, int n, double x)
static void GetFuncs1 (Optimizer1 o, int m0, int m1, double f)
static void GetGrads1 (Optimizer1 o, int m0, int m1, int n, int is, int js, double g)
static void GetContVars1 (Optimizer1 o, int n, double x)
static void SetBestContVars1 (Optimizer1 o, int n, double x)

getdist (RealVector &x1, RealVector &x2)
mindist (RealVector &x, RealMatrix &xset, int except)
mindistindx (RealVector &x, RealMatrix &xset, IntVector &indx)

getRmax (RealMatrix &xset)

Isfinite (const Real &x)

template<typename T> T abort_handler_t (int code)

start_grid_computing (char *analysis_driver_script, char *params_file, char *results_file)

stop_grid_computing ()

perform_analysis (char *iteration_num)

asstring (const T &val)

bool operator== (const MergedVariables &vars1, const MergedVariables &vars2)

equality operator

PACKBUF (int, MPI_INT)

UNPACKBUF (int, MPI_INT)

PACKSIZE (int, MPI_INT)

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

insert an int
- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_int &data)`
  insert a u_int

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const long &data)`
  insert a long

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_long &data)`
  insert a u_long

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const short &data)`
  insert a short

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_short &data)`
  insert a u_short

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const char &data)`
  insert a char

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const u_char &data)`
  insert a u_char

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const double &data)`
  insert a double

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const float &data)`
  insert a float

- `MPIPackBuffer & operator<< (MPIPackBuffer &buff, const bool &data)`
  insert a bool

- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, int &data)`
  extract an int

- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_int &data)`
  extract a u_int

- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, long &data)`
  extract a long

- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_long &data)`
  extract a u_long

- `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, short &data)`
  extract a short
- **MPIUnpackBuffer** & operator>>(MPIUnpackBuffer &buff, u_short &data)
  
  extract a u_short

- **MPIUnpackBuffer** & operator>>(MPIUnpackBuffer &buff, char &data)
  
  extract a char

- **MPIUnpackBuffer** & operator>>(MPIUnpackBuffer &buff, u_char &data)
  
  extract a u_char

- **MPIUnpackBuffer** & operator>>(MPIUnpackBuffer &buff, double &data)
  
  extract a double

- **MPIUnpackBuffer** & operator>>(MPIUnpackBuffer &buff, float &data)
  
  extract a float

- **MPIUnpackBuffer** & operator>>(MPIUnpackBuffer &buff, bool &data)
  
  extract a bool

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

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

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

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

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

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

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

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

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

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

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

- void dn2f_ (int n, int p, Real x, Calcrj, int *iv, int *lv, Real *v, int *ui, void *ur, Vf)
- void dn2fb_ (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 *, int *, Real *)
- double dr7mdc_ (int *)
- double rnum1 (void)
- double rnum2 (void)
- bool operator== (const ParamResponsePair &pair1, const ParamResponsePair &pair2)

  equality operator

- bool vars_set_compare (const ParamResponsePair &database_pr, const void *search_pr)

  search function for a particular ParamResponsePair within a List

- bool eval_id_compare (const ParamResponsePair &pair, const void *id)

  search function for a particular ParamResponsePair within a List

- bool eval_id_sort_fn (const ParamResponsePair &pr1, const ParamResponsePair &pr2)

  sort function for ParamResponsePair

- istream & operator>>(istream &, ParamResponsePair &pair)

  istream extraction operator for ParamResponsePair

- ostream & operator<<(ostream &, const ParamResponsePair &pair)

  ostream insertion operator for ParamResponsePair

- BiStream & operator>>(BiStream &, ParamResponsePair &pair)

  BiStream extraction operator for ParamResponsePair.

- BoStream & operator<<(BoStream &, const ParamResponsePair &pair)

  BoStream insertion operator for ParamResponsePair.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &, ParamResponsePair &pair)

  MPIUnpackBuffer extraction operator for ParamResponsePair.

- MPIPackBuffer & operator<<(MPIPackBuffer &, const ParamResponsePair &pair)

  MPIPackBuffer insertion operator for ParamResponsePair.

- bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)

  inequality operator
7.1 Dakota Namespace Reference

- 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

Variables

- ProblemDescDB dummy_db
  
  dummy ProblemDescDB object used for mandatory reference initialization when a real ProblemDescDB instance is unavailable

- ParallelLibrary dummy_lib
  
  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

- Graphics dakota_graphics
  
  the global Dakota::Graphics object used by strategies, models, and approximations

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

- **ParallelLibrary dummy_lib**
  dummy `ParallelLibrary` object used for mandatory reference initialization when a real `ParallelLibrary` instance is unavailable

- **Dakota_funcs * DF**
- **Dakota_funcs DakFuncs0**
  **ostream * dakota_cout = &cout**
  DAKOTA stdout initially points to cout, but may be redirected to a tagged ofstream if there are concurrent iterators.

  **ostream * dakota_cerr = &cerr**
  DAKOTA stderr initially points to cerr, but may be redirected to a tagged ofstream if there are concurrent iterators.

- **PRPList data_pairs**
  list of all parameter/response pairs.

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

- **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.C` overrides this default value)

- **ParallelLibrary dummy_lib (0)**
  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

- **ostream * dakota_cout**
  DAKOTA stdout initially points to cout, but may be redirected to a tagged ofstream if there are concurrent iterators.

- **ostream * dakota_cerr**
  DAKOTA stderr initially points to cerr, but may be redirected to a tagged ofstream if there are concurrent iterators.

- **int write_precision**
used in ostream data output functions (restart_util.C overrides this default value)

- `int mc_ptr_int`
  global pointer for ModelCenter API

- `FILE * yyin`
  - `const int LHS_NONRANDOM = 0`
  - `const int LHS_RANDOM = 1`
  - `Dakota::GSL_Singleton GSL_RNG`
  - `ParallelLibrary dummy_lib`
    dummy `ParallelLibrary` object used for mandatory reference initialization when a real `ParallelLibrary` instance is unavailable

- `const int LARGE_SCALE = 100`
- `const size_t _NPOS = ~(size_t)0`
  special value returned by index() when entry not found

### 7.1.1 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 *.h.

### 7.1.2 Function Documentation

#### 7.1.2.1 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.

#### 7.1.2.2 bool Dakota::operator== (const DistinctVariables & vars1, const DistinctVariables & vars2)

equality operator

Checks each array using operator== from data_types.C. Labels are ignored.

#### 7.1.2.3 Real Dakota::getdist (RealVector & x1, RealVector & x2)

Gets the Euclidean distance between x1 and x2
7.1.2.4 **Real Dakota::mindist (RealVector & x, 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.

7.1.2.5 **Real Dakota::mindistindx (RealVector & x, RealMatrix & xset, IntVector & indx)**

Gets the min distance between x and points in the set xset defined by the nindx values in indx.

7.1.2.6 **Real Dakota::getRmax (RealMatrix & xset)**

Gets the maximum of the min distance between each point and the rest of the set.

7.1.2.7 **string Dakota::asstring (const T & val)**

Creates a string from the argument val using an ostringstream.

This only gets used in this file and is only ever called with ints so no error checking is in place.

**Parameters:**
- **val** The value of type T to convert to a string.

**Returns:**
- The string representation of val created using an ostringstream.

7.1.2.8 **bool Dakota::vars_set_compare (const ParamResponsePair & database_pr, const void * search_pr)** [inline]

search function for a particular ParamResponsePair within a List

a global function to compare the parameter values, ASV, & interface id of a particular database_pr (presumed to be in the global history list) with a passed in set of parameters, ASV, & interface id provided by search_pr.

7.1.2.9 **bool Dakota::eval_id_compare (const ParamResponsePair & pair, const void * id)** [inline]

search function for a particular ParamResponsePair within a List

a global function to compare the evalId of a particular ParamResponsePair (from a List) with a passed in evaluation id. *(int*)id) construct casts void* to int* and then dereferences.

7.1.2.10 **bool Dakota::eval_id_sort_fn (const ParamResponsePair & pr1, const ParamResponsePair & pr2)** [inline]

sort function for ParamResponsePair
7.1.2.11  void print_restart (int argc, char ** argv, String print_dest)

print a restart file

Usage: "dakota_restart_util print dakota.rst"
"dakota_restart_util to_neutral dakota.rst dakota.neu"

Prints all 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.

7.1.2.12  void print_restart_tabular (int argc, char ** argv, String print_dest)

print a restart file (tabular format)

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/TECPLLOT data file).

7.1.2.13  void read_neutral (int argc, char ** argv)

read a restart file (neutral file format)

Usage: "dakota_restart_util from_neutral dakota.neu dakota.rst"

Reads evaluations from a neutral file. This is used for translating binary files between platforms.

7.1.2.14  void repair_restart (int argc, char ** argv, String identifier_type)

repair a restart file by removing corrupted evaluations

Usage: "dakota_restart_util remove 0.0 dakota_old.rst dakota_new.rst"
"dakota_restart_util remove_ids 2 7 13 dakota_old.rst dakota_new.rst"

Repairs a restart file by removing corrupted evaluations. The identifier for evaluation removal can be either a
double precision number (all evaluations having a matching response function value are removed) or a list of
integers (all evaluations with matching evaluation ids are removed).

7.1.2.15  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.
7.1.3 Variable Documentation

7.1.3.1 Dakota_funcs DakFuncs0

Initial value:

```c
{
  fprintf,
  abort_handler,
  dlsolver_option,
  continuous_lower_bounds1,
  continuous_upper_bounds1,
  nonlinear_ineq_constraint_lower_bounds1,
  nonlinear_ineq_constraint_upper_bounds1,
  nonlinear_eq_constraint_targets1,
  linear_ineq_constraint_lower_bounds1,
  linear_ineq_constraint_upper_bounds1,
  linear_eq_constraint_targets1,
  linear_ineq_constraint_coeffs1,
  linear_eq_constraint_coeffs1,
  bestFunctions1,
  ComputeResponses1,
  GetFuncs1,
  GetGrads1,
  GetContVars1,
  SetBestContVars1
}
```
7.2 SIM Namespace Reference

plug facilities into DAKOTA.

Classes

- class DirectFnApplicInterface
  
  Sample derived interface class for testing plug-ins using assign_rep().

7.2.1 Detailed Description

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.
8.1 ActiveSet Class Reference

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 (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*

- **const ShortArray & request_vector () const**
  
  *return the request vector*
void request_vector (const ShortArray &rv)
    set the request vector

void request_values (const int rv_val)
    set all request vector values

void request_value (const size_t index, const int rv_val)
    set the value of an entry in the request vector

const IntArray & derivative_vector () const
    return the derivative variables vector

void derivative_vector (const IntArray &dvv)
    set the derivative variables vector

void derivative_start_value (const int dvv_start_val)
    set the derivative variables vector values

void read (istream &s)
    read an active set object from an istream

void write (ostream &s) const
    write an active set object to an ostream

void write_annotated (ostream &s) const
    write an active set object to an 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

void write (MPIPackBuffer &s) const
    write an active set object to a packed MPI buffer

Private Attributes

- ShortArray requestVector
  the vector of response requests

- IntArray derivVarsVector
  the vector of variable ids used for computing derivatives
8.1 ActiveSet Class Reference

Friends

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

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

8.1.1 Detailed Description

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).

8.1.2 Member Data Documentation

8.1.2.1 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.

8.1.2.2 IntArray 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. The documentation for this class was generated from the following files:

- DakotaActiveSet.H
- DakotaActiveSet.C
8.2 AllConstraints Class Reference

employs the all data view.

Inheritance diagram for AllConstraints:

```
    Constraints                  VariablesUtil
                                  `                        `                  `  
                                  ¦                        ¦                   ¦
                                  ¦                        ¦                   ¦
                                  V                        V                   V
                      AllConstraints
```

Public Member Functions

- **AllConstraints ()**
  *default constructor*

- **AllConstraints (const ProblemDescDB &problem_db, const pair< short, short >&view)**
  *standard constructor*

- **~AllConstraints ()**
  *destructor*

- **const RealVector & continuous_lower_bounds () const**
  *return the active continuous variable lower bounds*

- **void continuous_lower_bounds (const RealVector &c_l_bnds)**
  *set the active continuous variable lower bounds*

- **const RealVector & continuous_upper_bounds () const**
  *return the active continuous variable upper bounds*

- **void continuous_upper_bounds (const RealVector &c_u_bnds)**
  *set the active continuous variable upper bounds*

- **const IntVector & discrete_lower_bounds () const**
  *return the active discrete variable lower bounds*

- **void discrete_lower_bounds (const IntVector &d_l_bnds)**
  *set the active discrete variable lower bounds*

- **const IntVector & discrete_upper_bounds () const**
  *return the active discrete variable upper bounds*
- **void discrete_upper_bounds (const IntVector &d_u_bnds)**
  set the active discrete variable upper bounds

- **RealVector all_continuous_lower_bounds () const**
  returns a single array with all continuous lower bounds

- **void all_continuous_lower_bounds (const RealVector &a_c_l_bnds)**
  sets all continuous lower bounds using a single array

- **RealVector all_continuous_upper_bounds () const**
  returns a single array with all continuous upper bounds

- **void all_continuous_upper_bounds (const RealVector &a_c_u_bnds)**
  sets all continuous upper bounds using a single array

- **IntVector all_discrete_lower_bounds () const**
  returns a single array with all discrete lower bounds

- **void all_discrete_lower_bounds (const IntVector &a_d_l_bnds)**
  sets all discrete lower bounds using a single array

- **IntVector all_discrete_upper_bounds () const**
  returns a single array with all discrete upper bounds

- **void all_discrete_upper_bounds (const IntVector &a_d_u_bnds)**
  sets all discrete upper bounds using a single array

- **void write (ostream &s) const**
  write a variable constraints object to an ostream

- **void read (istream &s)**
  read a variable constraints object from an istream

### Protected Member Functions

- **void copy_rep (const Constraints *con_rep)**
  Used by copy() to copy the contents of a letter class.

- **void reshape_rep (const Sizet2DArray &vars_comps)**
  Used by reshape(Sizet2DArray&) to reshape the contents of a letter class.
Private Attributes

- **RealVector allContinuousLowerBnds**
  uncertain, and continuous state variable types (all view).

- **RealVector allContinuousUpperBnds**
  uncertain, and continuous state variable types (all view).

- **IntVector allDiscreteLowerBnds**
  discrete state variable types (all view).

- **IntVector allDiscreteUpperBnds**
  discrete state variable types (all view).

- **size_t numCDV**
  number of continuous design variables

- **size_t numDDV**
  number of discrete design variables

### 8.2.1 Detailed Description

employs the all data view.

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The **AllConstraints** derived class combines design, uncertain, and state variable types but separates continuous and discrete domain types. The result is combined continuous bounds arrays (allContinuousLowerBnds, allContinuousUpperBnds) and combined discrete bounds arrays (allDiscreteLowerBnds, allDiscreteUpperBnds). Parameter and DACE studies currently use this approach (see Variables::get_variables(problem_db) for variables view selection; variables view is passed to the Constraints constructor in Model).

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 **AllConstraints** (const ProblemDescDB & problem_db, const pair< short, short > & view)

standard constructor

In this class, the all data approach (design, uncertain, and state types are combined) is used. Iterators/strategies which use this class include: parameter studies, dace, and nond_sampling in all_variables mode. Extract fundamental lower and upper bounds and combine them into allContinuousLowerBnds, allContinuousUpperBnds, allDiscreteLowerBnds, and allDiscreteUpperBnds using utilities from VariablesUtil.

The documentation for this class was generated from the following files:
8.2 AllConstraints Class Reference

- AllConstraints.H
- AllConstraints.C
# AllVariables Class Reference

Inheritance diagram for AllVariables:

```
Variables | VariablesUtil
----------|--------------
          | AllVariables
```

## Public Member Functions

- `AllVariables ()`
  - *default constructor*

- `AllVariables (const ProblemDescDB &problem_db, const pair< short, short > &view)`
  - *standard constructor*

- `~AllVariables ()`
  - *destructor*

- `size_t tv () const`
  - *Returns total number of vars.*

- `const RealVector & continuous_variables () const`
  - *return the active continuous variables*

- `void continuous_variables (const RealVector &c_vars)`
  - *set the active continuous variables*

- `const IntVector & discrete_variables () const`
  - *return the active discrete variables*

- `void discrete_variables (const IntVector &d_vars)`
  - *set the active discrete variables*

- `const StringArray & continuous_variable_labels () const`
  - *return the active continuous variable labels*

- `void continuous_variable_labels (const StringArray &cv_labels)`
  - *set the active continuous variable labels*
const StringArray & discrete_variable_labels () const
   return the active discrete variable labels

void discrete_variable_labels (const StringArray &dv_labels)
   set the active discrete variable labels

size_t acv () const
   returns total number of continuous vars

size_t adv () const
   returns total number of discrete vars

RealVector all_continuous_variables () const
   returns a single array with all continuous variables

void all_continuous_variables (const RealVector &a_c_vars)
   sets all continuous variables using a single array

IntVector all_discrete_variables () const
   returns a single array with all discrete variables

void all_discrete_variables (const IntVector &a_d_vars)
   sets all discrete variables using a single array

StringArray all_continuous_variable_labels () const
   returns a single array with all continuous variable labels

void all_continuous_variable_labels (const StringArray &a_c_v_labels)
   sets all continuous variable labels using a single array

StringArray all_discrete_variable_labels () const
   returns a single array with all discrete variable labels

void all_discrete_variable_labels (const StringArray &a_d_v_labels)
   sets all discrete variable labels using a single array

StringArray all_variable_labels () const
   returns a single array with all variable labels

void read (istream &s)
   read a variables object from an istream

void write (ostream &s) const
   write a variables object to an ostream
- void write_aprepro (ostream &s) const
  write a variables object to an ostream in a Prepro format

- void read.annotated (istream &s)
  read a variables object in annotated format from an istream

- void write.annotated (ostream &s) const
  write a variables object in annotated format to an ostream

- void write_tabular (ostream &s) const
  write a variables object in tabular format to an ostream

- void read (BiStream &s)
  read a variables object from the binary restart stream

- void write (BoStream &s) const
  write a variables object to the binary restart stream

- void read (MPIUnpackBuffer &s)
  read a variables object from a packed MPI buffer

- void write (MPIPackBuffer &s) const
  write a variables object to a packed MPI buffer

Protected Member Functions

- void copy_rep (const Variables *vars_rep)
  Used by copy() to copy the contents of a letter class.

- void reshape_rep (const Sizet2DArray &vars_comps)
  Used by reshape() to reshape the contents of a letter class.

Private Member Functions

- void build_types_ids ()
  construct VarTypes and VarIds arrays using variables Components

Private Attributes

- RealVector allContinuousVars
  (design, uncertain, and state).
8.3 AllVariables Class Reference

- **IntVector allDiscreteVars**
  
  (design and state).

- **StringArray allContinuousLabels**
  
  (design, uncertain, and state).

- **StringArray allDiscreteLabels**
  
  (design and state).

**Friends**

- `bool operator==(const AllVariables &vars1, const AllVariables &vars2)`

  equality operator

8.3.1 Detailed Description

the all data view.

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The AllVariables derived class combines design, uncertain, and state variable types but separates continuous and discrete domain types. The result is a single array of continuous variables (allContinuousVars) and a single array of discrete variables (allDiscreteVars). Parameter and DACE studies currently use this approach (see Variables::get_variables(problem_db)).

8.3.2 Constructor & Destructor Documentation

8.3.2.1 AllVariables (const ProblemDescDB & problem_db, const pair< short, short > & view)

standard constructor

In this class, the all data approach (design, uncertain, and state types are combined) is used. Iterators/strategies which use this class include: parameter studies, DACE, and the all_variables mode of nond_sampling. Extract fundamental variable types and labels and combine them into allContinuousVars, allDiscreteVars, allContinuousLabels, and allDiscreteLabels using utilities from VariablesUtil.

The documentation for this class was generated from the following files:

- AllVariables.H
- AllVariables.C
8.4 AnalysisCode Class Reference

processes for managing simulations.

Inheritance diagram for AnalysisCode:

```
AnalysisCode
   "
ForkAnalysisCode  SysCallAnalysisCode
```

Public Member Functions

- void **define_filenames** (const int id)  
  *file and tagging options*

- void **write_parameters_files** (const Variables &vars, const ActiveSet &set, const int id)  
  *write_parameters_file() in either standard or aprepro format*

- void **read_results_files** (Response &response, const int id)  
  *read the response object from one or more results files*

- const StringArray & **program_names** () const  
  *return programNames*

- const String & **input_filter_name** () const  
  *return iFilterName*

- const String & **output_filter_name** () const  
  *return oFilterName*

- const String & **parameters_filename** () const  
  *return paramsFileName*

- const String & **results_filename** () const  
  *return resultsFileName*

- const String & **results_filename** (const int id)  
  *return the results filename entry in fileNameMap corresponding to id*

- void **suppress_output_flag** (const bool flag)  
  *set suppressOutputFlag*
protected member functions

- bool suppress_output_flag () const
  
  return suppressOutputFlag

- bool multiple_parameters_filenames () const
  
  return multipleParamsFiles

protected attributes

- bool suppressOutputFlag
  
  flag set by master processor to suppress output from slave processors

- bool verboseFlag
  
  flag for additional analysis code output if method verbosity is set

- bool fileTagFlag
  
  flags tagging of parameter/results files

- bool fileSaveFlag
  
  flags retention of parameter/results files

- bool aprereproFlag
  
  format for parameter files

- bool multipleParamsFiles
  
  analysis drivers

- String iFilterName
  
  the name of the input filter (input_filter user specification)

- String oFilterName
  
  the name of the output filter (output_filter user specification)

- StringArray programNames
  
  specification
• size_t numPrograms
  
  the number of analysis code programs (length of programNames)

• String specifiedParamsFileName
  
  the name of the parameters file from user specification

• String paramsFileName
  
  temp files)

• String specifiedResultsFileName
  
  the name of the results file from user specification

• String resultsFileName
  
  the results file name actually used (modified with tagging or temp files)

• map<int, pair<String, String>> fileNameMap
  
  evaluations. Map key is the function evaluation identifier.

Private Member Functions

• void write_parameters_file (const Variables &vars, const ActiveSet &set, const StringArray &an_comps, const String &params_fname)
  
  standard or aprepro format

Private Attributes

• ParallelLibrary & parallelLib
  
  reference to the ParallelLibrary object. Used in define_filenames().

• String2DArray analysisComponents
  
  (from the analysis_components interface specification)

8.4.1 Detailed Description

processes for managing simulations.

The AnalysisCode class hierarchy provides simulation spawning services for ApplicationInterface derived classes and alleviates these classes of some of the specifics of simulation code management. The hierarchy does not employ the letter-envelope technique since the ApplicationInterface derived classes instantiate the appropriate derived AnalysisCode class directly.

The documentation for this class was generated from the following files:
- AnalysisCode.H
- AnalysisCode.C
8.5 Analyzer Class Reference

hierarchy.

Inheritance diagram for Analyzer::

```
    Iterator
     |
    Analyzer

NonD
    |
    NonDEvidence
    |
    NonDIntegration
    |
    NonDPolynomialChaos
    |
    NonDReliability
    |
    NonDSampling

PStudyDACE
    |
    DDACEDesignCompExp
    |
    FSUDesignCompExp
    |
    ParamStudy
    |
    PSUADEDesignCompExp
```

Public Member Functions

- `const VariablesArray & all_variables () const`
  `return the complete set of evaluated variables`

- `const ResponseArray & all_responses () const`
  `return the complete set of computed responses`

- `const VariablesArray & variables_array_results () const`
  `return multiple final iterator solutions (variables)`

- `const ResponseArray & response_array_results () const`
  `return multiple final iterator solutions (response)`

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 update_best (const RealVector &vars, const Response &response, const int eval_num)**
  
  *compares current evaluation to best evaluation and updates best*

- **virtual void vary_pattern (bool pattern_flag)**
  
  *sets varyPattern in derived classes that support it*

- **virtual void get_parameter_sets (const Model &model)**
  
  *Returns one block of samples (ndim * num_samples).*

- **void evaluateParameter_sets (Model &model, bool log_resp_flag, bool log_best_flag)**
  
  *into response sets (allResponses)*

- **void var_based_decomp (const int ndim, const int num_samples)**

- **void volumetric_quality (int ndim, int num_samples, double *sample_points)**
  
  *Calculation of volumetric quality measures.*

- **void print_vbd (ostream &s, const RealVectorArray &S, const RealVectorArray &T) const**
  
  *Printing of VBD results.*

### Protected Attributes

- **VariablesArray allVariables**
  
  *array of all variables evaluated*

- **ResponseArray allResponses**
  
  *array of all responses computed*

- **StringArray allHeaders**
  
  *array of headers to insert into output while evaluating allVariables*

- **bool qualityFlag**
flag to indicated if quality metrics were calculated

- double chiMeas
  quality measure
- double dMeas
  quality measure
- double hMeas
  quality measure
- double tauMeas
  quality measure

8.5.1 Detailed Description

hierarchy.

The Analyzer class provides common data and functionality for various types of systems analysis, including nondeterministic analysis, design of experiments, and parameter studies.

8.5.2 Member Function Documentation

8.5.2.1 void evaluate_parameter_sets (Model & model, bool log_resp_flag, bool log_best_flag) [protected]

into response sets (allResponses)

Convenience function for derived classes with sets of function evaluations to perform (e.g., NonDSampling, DDACEDesignCompExp, FSUDesignCompExp, ParamStudy).

8.5.2.2 void var_based_decomp (const int ndim, const 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.

8.5.2.3 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.
8.5 Analyzer Class Reference

8.5.2.4 void print_vbd (ostream & s, const RealVectorArray & S, const RealVectorArray & T) const

[protected]

Printing of VBD results.

printing of variance based decomposition indices.

The documentation for this class was generated from the following files:

- DakotaAnalyzer.H
- DakotaAnalyzer.C
8.6 ApplicationInterface Class Reference

interfaces to simulation codes.

Inheritance diagram for ApplicationInterface:

```
        Interface
         
    ApplicationInterface
     
DirectFnApplicInterface     ForkApplicInterface     GridApplicInterface     SysCallApplicInterface
```

Public Member Functions

- `ApplicationInterface(const ProblemDescDB &problem_db)`
  
  constructor

- `~ApplicationInterface()`
  
  destructor

Protected Member Functions

- `void init_communicators(const IntArray &message_lengths, const int &max_iterator_concurrency)`
  iterator and concurrent multiprocessor analyses within an evaluation.

- `void set_communicators(const IntArray &message_lengths)`
  (the partitions are already allocated in ParallelLibrary).

- `void free_communicators()`
  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
• void map (const Variables &vars, const ActiveSet &set, Response &response, const bool asynch_flag=false)
  
  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 ResponseArray & synch ()
  
  the beforeSynchCorePRPList queue and returns all jobs

• const IntResponseMap & synch_nowait ()
  
  beforeSynchCorePRPList queue and returns a partial list 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

• virtual void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  
  that is specific to a derived class.

• virtual void derived_map_asynch (const ParamResponsePair &pair)
  
  asynchronous evaluation that is specific to a derived class.

• virtual void derived_synch (PRPList &prp_list)
  
  classes. This version waits for at least one completion.

• virtual void derived_synch_nowait (PRPList &prp_list)
  
  any completions if none are immediately available.

• void self_schedule_analyses ()
  
  evaluation using message passing

• void serve_analyses_synch ()
  
  analysis job at a time

• virtual int derived_synchronous_local_analysis (const int &analysis_id)
  
  ApplicationInterface::serve_analyses_synch().
Protected Attributes

- `ParallelLibrary & parallelLib`
  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`
  number of analysis servers

- `bool multiProcAnalysisFlag`
  flag for multiprocessor analysis partitions

- `bool asynchLocalAnalysisFlag`
  flag for asynchronous local parallelism of analyses

- `int asynchLocalAnalysisConcurrency`
  scheduling and specifies hybrid concurrency when message passing

- `int numAnalysisDrivers`
  (from the analysis_drivers interface specification)

- `IntSet completionSet`
  and `derived_synch_nowait()`
Private Member Functions

- bool duplication_detect (const Variables &vars, Response &response, const bool asynch_flag)
  evaluation request has already been performed or queued

- void self_schedule_evaluations ()
  using message passing; executes on iteratorComm master

- void static_schedule_evaluations ()
  using message passing; executes on iteratorComm master

- void asynchronous_local_evaluations (PRPList &prp_list)
  the local processor

- void synchronous_local_evaluations (PRPList &prp_list)
  the local processor

- void asynchronous_local_evaluations_nowait (PRPList &prp_list)
  process any completed jobs

- void serve_evaluations_synch ()
  one synchronous evaluation at a time

- void serve_evaluations_asynch ()
  multiple asynchronous evaluations

- void serve_evaluations_peer ()
  one synchronous evaluation at a time as part of the 1st peer

- void set_evaluation_communicators (const IntArray &message_lengths)
  following ParallelLibrary::init_evaluation_communicators().

- void set_analysis_communicators ()
  following ParallelLibrary::init_analysis_communicators().

- void check_configuration (const int &max_iterator_concurrency)
  perform some error checks on the parallel configuration

- const ParamResponsePair & get_source_pair (const Variables &target_vars)
  evaluation to the failed "target"

- void continuation (const Variables &target_vars, const ActiveSet &set, Response &response, const ParamResponsePair &source_pair, int failed_eval_id)
  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
  number of evaluation servers

- bool eaMessagePass
  flag for message passing at ea scheduling level

- int procsPerAnalysis
  processors per analysis servers

- int lenVarsMessage
  computed in Model::init_communicators()

- int lenVarsActSetMessage
  ActiveSet object; computed in Model::init_communicators().

- int lenResponseMessage
  computed in Model::init_communicators()

- int lenPRPairMessage
  computed in Model::init_communicators()

- String evalScheduling
  auto-configure logic in ParallelLibrary::resolve_inputs().
8.6 Application Interface Class Reference

- **String analysisScheduling**
  
  *auto-configure logic in ParallelLibrary::resolve_inputs()*

- **int asynchLocalEvalConcurrency**
  
  *scheduling and specifies hybrid concurrency when message passing*

- **String interfaceSynchronization**
  
  *or asynchronous*

- **bool headerFlag**
  
  *function may be called many times prior to any completions*

- **bool asvControlFlag**
  
  *on each evaluation.*

- **bool evalCacheFlag**
  
  *cache (i.e., queries and insertions using the data_pairs list).*

- **bool restartFileFlag**
  
  *insertions into write_restart.*

- **ShortArray defaultASV**
  
  *the static ASV values used when the user has selected asvControl = off*

- **String failAction**
  
  *retry, recover, or continuation*

- **int failRetryLimit**
  
  *limit on the number of retries for the retry failAction*

- **RealVector failRecoveryFnVals**
  
  *the dummy function values used for the recover failAction*

- **IntList beforeSynchIdList**
  
  *bookkeeps fnEvalId's of _all_ asynchronous evaluations (new & duplicate)*

- **IntResponseMap historyDuplicateMap**
  
  *evaluations. Map key is fnEvalId, mad data is corresponding response.*

- **std::map< int, pair< PRPLIter, Response >> beforeSynchDuplicateMap**
  
  *beforeSynchCorePRPList evaluations*

- **PRPList beforeSynchCorePRPList**
  
  *that is later scheduled in synch() or synch_nowait().*
- **PRPList beforeSynchAlgPRPList**
  
  *that is later evaluated in synch() or synch_nowait().*

- **ResponseList beforeSynchTotalRespList**
  
  *asynchronous map() and later used in synch() or synch_nowait().*

- **IntSet runningSet**
  
  *used by asynchronous_local_nowait to bookkeep which jobs are running*

### 8.6.1 Detailed Description

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.

### 8.6.2 Member Function Documentation

#### 8.6.2.1 void init_serial () [inline, protected, virtual]

DataInterface.C 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 `DirectFnApplicInterface::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`.

#### 8.6.2.2 void map (const Variables & vars, const ActiveSet & set, Response & response, const bool asynch_flag = false) [protected, virtual]

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 beforeSynchCorePRPList 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`. 
8.6.2.3 const ResponseArray & synch () [protected, virtual]

The beforeSynchronCorePRPList queue 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_synch() in derived Model classes.

Reimplemented from Interface.

8.6.2.4 const IntResponseMap & synch_nowait () [protected, virtual]

beforeSynchronCorePRPList queue and returns a partial list of completed jobs

This function will eventually provide nonblocking synchronization for all cases of asynchronous evaluations, however it currently supports only the local asynchronous case since nonblocking message passing schedulers have not yet been implemented. Called from derived_synch_nowait() in derived Model classes.

Reimplemented from Interface.

8.6.2.5 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_asynch(), serve_evaluations_peer(), or serve_evaluations_synch() according to specified concurrency and self/static scheduler configuration.

Reimplemented from Interface.

8.6.2.6 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.

8.6.2.7 void self_schedule_analyses () [protected]

evaluation using message passing

This code is called from derived classes to provide the master portion of a master-slave algorithm for the dynamic self-scheduling of analyses among slave servers. It is patterned after self_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. Self-scheduling approach 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.

**8.6.2.8 void serve_analyses_synch () [protected]**

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()`.

**8.6.2.9 bool duplication_detect (const `Variables` & vars, `Response` & response, const bool asynch_flag) [private]**

evaluation request has already been performed or queued

Called from `map()` to check incoming evaluation request for duplication with content of data_pairs and before-SynchCorePRPList. If duplication is detected, return true, else return false. Manage bookkeeping with history-DuplicateMap 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 beforeSynchCorePRPList search would still be meaningful. Since the intent of this request is to streamline operations, both list searches are bypassed.

**8.6.2.10 void self_schedule_evaluations () [private]**

using message passing; executes on iteratorComm master

This code is called from `synch()` to provide the master portion of a master-slave algorithm for the dynamic self-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. Self-scheduling approach 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-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

**8.6.2.11 void static_schedule_evaluations () [private]**

using message passing; executes on iteratorComm master

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from `synch()` in order to assign a static schedule. It matches `serve_evaluations_peer()` for any other processors within the 1st evaluation partition and `serve_evaluations_synch()/serve_evaluations_asynch()` for all other evaluation partitions (depending on asynchLocalEvalConcurrency). It performs function evaluations locally for its portion of the static schedule 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 beforeSynchCorePRPList (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.

8.6.2.12 \textbf{void asynchronous\_local\_evaluations (PRPList \& prp\_list) [private]}

the local processor

This function provides blocking synchronization for the local asynch case (background system call, non-blocking fork, or threads). It can be called from \texttt{synch()} for a complete local scheduling of all asynchronous jobs or from \texttt{static\_schedule\_evaluations()} to perform a local portion of the total job set. It uses the \texttt{derived\_map\_asynch()} to initiate asynchronous evaluations and \texttt{derived\_synch()} to capture completed jobs, and mirrors the \texttt{self\_schedule\_evaluations()} message passing scheduler as much as possible (\texttt{derived\_synch()} is modeled after MPI\_Waitsome()).

8.6.2.13 \textbf{void synchronous\_local\_evaluations (PRPList \& prp\_list) [private]}

the local processor

This function provides blocking synchronization for the local synchronous case (foreground system call, blocking fork, or procedure call from \texttt{derived\_map()}). It is called from \texttt{static\_schedule\_evaluations()} to perform a local portion of the total job set.

8.6.2.14 \textbf{void asynchronous\_local\_evaluations\_nowait (PRPList \& prp\_list) [private]}

process any completed jobs

This function provides nonblocking synchronization for the local asynch case (background system call, non-blocking fork, or threads). It is called from \texttt{synch\_nowait()} and passed the complete set of all asynchronous jobs (beforeSynchCorePRPList). It uses \texttt{derived\_map\_asynch()} to initiate asynchronous evaluations and \texttt{derived\_synch\_nowait()} to capture completed jobs in nonblocking mode. It mirrors a nonblocking message passing scheduler as much as possible (\texttt{derived\_synch\_nowait()} modeled after MPI\_Testsome()). The result of this function is \texttt{rawResponseMap}, which uses \texttt{fn\_eval\_id} as a key. It is assumed that the incoming \texttt{prp\_list} contains only active and new jobs - i.e., all completed jobs are cleared by \texttt{synch\_nowait()}.

8.6.2.15 \textbf{void serve\_evaluations\_synch () [private]}

one synchronous evaluation at a time

This code is invoked by \texttt{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 \texttt{stop\_evaluation\_servers()}).

8.6.2.16 \textbf{void serve\_evaluations\_asynch () [private]}

multiple asynchronous evaluations

This code is invoked by \texttt{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).

8.6.2.17 void serve_evaluations_peer () [private]

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 static_schedule_evaluations(), or the bcast() in map().

The documentation for this class was generated from the following files:

- ApplicationInterface.H
- ApplicationInterface.C
8.7 Approximation Class Reference

Base class for the approximation class hierarchy.

Inheritance diagram for Approximation:

Public Member Functions

- **Approximation ()**
  
  *default constructor*

- **Approximation (ProblemDescDB &problem_db, const size_t &num_vars)**
  
  *standard constructor for envelope*

- **Approximation (const String &approx_type, short approx_order, const size_t &num_vars)**
  
  *alternate constructor*

- **Approximation (const Approximation &approx)**
  
  *copy constructor*

- **virtual ~Approximation ()**
  
  *destructor*

- **Approximation operator= (const Approximation &approx)**
  
  *assignment operator*

- **virtual const Real & get_value (const RealVector &x)**
  
  *retrieve the approximate function value for a given parameter vector*

- **virtual const RealBaseVector & get_gradient (const RealVector &x)**
  
  *retrieve the approximate function gradient for a given parameter vector*

- **virtual const RealMatrix & get_hessian (const RealVector &x)**
  
  *retrieve the approximate function Hessian for a given parameter vector*

- **virtual const Real & get_variance (const RealVector &x)**
  
  *retrieve the variance of the predicted value for a given parameter vector*
virtual const RealVector & approximation_coefficients () const
    return the coefficient array computed by find_coefficients()

virtual void approximation_coefficients (const RealVector &approx_coeffs)
    computing with find_coefficients()

virtual void print_coefficients (ostream &s) const
    print the coefficient array computed in find_coefficients()

virtual int num_coefficients () const
    derived class approximation type in numVars dimensions

virtual int num_constraints () const
    return the number of constraints to be enforced via anchorPoint

virtual void clear_current ()
    clear current build data in preparation for next build

int required_samples (bool constraint_flag) const
    type in numVars dimensions. Uses num_coefficients() and num_constraints().

int num_variables () const
    return the number of variables used in the approximation

const List< SurrogateDataPoint > & current_points () const
    return currentPoints

const SurrogateDataPoint & anchor_point () const
    return anchorPoint

void update (const Variables &vars, const Response &response, const int &fn_index)
    populates/replaces anchorPoint

void update (const RealVector &c_vars, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)
    populates/replaces anchorPoint

void update (const VariablesArray &vars_array, const ResponseArray &resp_array, const int &fn_index)
    populates/replaces currentPoints

void append (const Variables &vars, const Response &response, const int &fn_index)
    appends one additional entry to currentPoints

void append (const RealVector &c_vars, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)
    appends one additional entry to currentPoints
void append (const VariablesArray &vars_array, const ResponseArray &resp_array, const int &fn_index)
  appends multiple additional entries to currentPoints

void build ()
  builds the approximation by invoking find_coefficients()

bool anchor () const
  queries the status of anchorPoint

void clear_all ()
  clear all build data (current and history) to restore original state

void set_bounds (const RealVector &lower, const RealVector &upper)
  set approximation lower and upper bounds (currently only used by graphics)

void draw_surface ()
  problems only)

Approximation * approx_rep () const
  that are not mapped to the top Approximation level

Protected Member Functions

Approximation (BaseConstructor, ProblemDescDB &problem_db, const size_t &num_vars)
  derived class constructors - Coplien, p. 139)

virtual void find_coefficients ()
  calculate the data fit coefficients using currentPoints and anchorPoint

Protected Attributes

bool useGradsFlag
  trust region), but not require gradient evaluations at every point.

bool verboseFlag
  flag for verbose approximation output

int numVars
  number of variables in the approximation

String approxType
  approximation type identifier
- short approxOrder
  orthogonal polynomials, and Taylor series

- Real approxValue
  value of the approximation returned by get_value()

- RealBaseVector approxGradient
  gradient of the approximation returned by get_gradient()

- RealMatrix approxHessian
  Hessian of the approximation returned by get_hessian().

- Real approxVariance
  value of the approximation returned by get_variance()

- List< SurrogateDataPoint > currentPoints
  are fit approximately (e.g., using least squares regression).

- SurrogateDataPoint anchorPoint
  least squares regression).

Private Member Functions

- Approximation * get_approx (ProblemDescDB &problem_db, const size_t &num_vars)
  approxRep to the appropriate derived type.

- Approximation * get_approx (const String &approx_type, short approx_order, const size_t &num_vars)
  approxRep to the appropriate derived type.

- void add (const Variables &vars, const Response &response, const int &fn_index, bool anchor_flag)
  add().

- void add_point (const RealVector &x, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)
  add a new data point by appending to currentPoints

- void add_anchor (const RealVector &x, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)
  add a new data point by assigning to anchorPoint
8.7 Approximation Class Reference

Private Attributes

- **RealVector approxLowerBounds**
  
  approximation lower bounds (used only by 3D graphics)

- **RealVector approxUpperBounds**
  
  approximation upper bounds (used only by 3D graphics)

- **Approximation * approxRep**
  
  pointer to the letter (initialized only for the envelope)

- **int referenceCount**
  
  number of objects sharing approxRep

8.7.1 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_approximation()) serves as the letter.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 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 Approximation object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

8.7.2.2 Approximation (ProblemDescDB & problem_db, const 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.
8.7.2.3 **Approximation** (const `String & approx_type, short approx_order, const size_t & num_vars)

alternate constructor

This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to `problem_db`, the letter class is not fully populated. This constructor executes `get_approx(type)`, which invokes the default constructor of the derived letter class, which in turn invokes the default constructor of the base class.

8.7.2.4 **Approximation** (const `Approximation & approx)

copy constructor

Copy constructor manages sharing of approxRep and incrementing of referenceCount.

8.7.2.5 ~**Approximation** () [virtual]

destructor

Destructor decrements referenceCount and only deletes approxRep when referenceCount reaches zero.

8.7.2.6 **Approximation** (BaseConstructor, ProblemDescDB & `problem_db, const size_t & num_vars) [protected]

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).

8.7.3 Member Function Documentation

8.7.3.1 **Approximation** operator= (const `Approximation & approx)

assignment operator


8.7.3.2 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`. 
8.7.3.3 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).

8.7.3.4 Approximation * get_approx (ProblemDescDB & problem_db, const size_t & num_vars) [private]

approxRep to the appropriate derived type.
Used only by the envelope constructor to initialize approxRep to the appropriate derived type.

8.7.3.5 Approximation * get_approx (const String & approx_type, short approx_order, const size_t & num_vars) [private]

approxRep to the appropriate derived type.
Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
The documentation for this class was generated from the following files:

- DakotaApproximation.H
- DakotaApproximation.C
8.8 ApproximationInterface Class Reference

approximations to simulation-based results.

Inheritance diagram for ApproximationInterface::

```
  Interface
  |___ ApproximationInterface
```

Public Member Functions

- **ApproximationInterface** (ProblemDescDB &problem_db, const Variables &actual_model_vars, const size_t &num_fns)
  
  *primary constructor*

- **ApproximationInterface** (const String &approx_type, const short &approx_order, const Variables &actual_model_vars, const size_t &num_fns)
  
  *alternate constructor for instantiations on the fly*

- ~ApproximationInterface()
  
  *destructor*

Protected Member Functions

- void map (const Variables &vars, const ActiveSet &set, Response &response, const bool asynch_flag=false)
  
  *the variables to the responses using functionSurfaces*

- int minimum_samples (bool constraint_flag) const
  
  *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 Response &response)
- void update_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array)
- void append_approximation (const Variables &vars, const Response &response)
- void append_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array)
- void build_approximation (const RealVector &lower_bnds, const RealVector &upper_bnds)
• void clear_current ()
  clears current data from an approximation interface

• void clear_all ()
  clears all data from an approximation interface

• bool anchor () const
  queries the presence of an anchorPoint within an approximation interface

• Array< Approximation > & approximations ()
  retrieve the Approximations within an ApproximationInterface

• const RealVectorArray & approximation_coefficients ()
  within an ApproximationInterface

• void approximation_coefficients (const RealVectorArray &approx_coeffs)
  within an ApproximationInterface

• void print_coefficients (ostream &s, size_t index) const
  Approximation instance within an ApproximationInterface.

• const RealVector & approximation_variances (const RealVector &c_vars)
  within an ApproximationInterface

• const List< SurrogateDataPoint > & approximation_data (size_t index)
  within an ApproximationInterface

• const ResponseArray & synch ()
  recovers data from a series of asynchronous evaluations (blocking)

• const IntResponseMap & synch_nowait ()
  recovers data from a series of asynchronous evaluations (nonblocking)

Private Attributes

• IntSet approxFnIndices
  response function subset that is approximated

• Array< Approximation > functionSurfaces
  list of approximations, one per response function

• RealVectorArray functionSurfaceCoeffs
  response function

• RealVector functionSurfaceVariances
List< SurrogateDataPoint > functionSurfaceDataPoints
for a particular response function

bool graphicsFlag
controls 3D graphics of approximation surfaces

Variables actualModelVars
among differing variable views

IntResponseMap beforeSynchResponseMap
but asynchronous virtual functions are supported through bookkeeping.

8.8.1 Detailed Description

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.

8.8.2 Member Function Documentation

8.8.2.1 void update_approximation (const Variables & vars, const Response & response) [protected, virtual]
This function populates/replaces each Approximation::anchorPoint with the incoming variables/response data
point.
Reimplemented from Interface.

8.8.2.2 void update_approximation (const VariablesArray & vars_array, const ResponseArray &
resp_array) [protected, virtual]
This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.
Reimplemented from Interface.

8.8.2.3 void append_approximation (const Variables & vars, const Response & response) [protected, virtual]
This function appends to each Approximation::currentPoints with one incoming variables/response data point.
Reimplemented from Interface.
8.8.2.4  **void append_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array)**  [protected, virtual]

This function appends to each Approximation::currentPoints with multiple incoming variables/response data points.
Reimplemented from Interface.

8.8.2.5  **void build_approximation (const RealVector & lower_bnds, const RealVector & upper_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.

8.8.3  **Member Data Documentation**

8.8.3.1  **Array<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.

The documentation for this class was generated from the following files:

- ApproximationInterface.H
- ApproximationInterface.C
8.9 Array Class Template Reference

Template class for the Dakota bookkeeping array.

Public Member Functions

- `Array ()`
  *Default constructor.*

- `Array (size_t size)`
  *Constructor which takes an initial size.*

- `Array (size_t size, const T &initial_val)`
  *Constructor which takes an initial size and an initial value.*

- `Array (const Array<T> &a)`
  *Copy constructor.*

- `Array (const T *p, size_t size)`
  *Constructor which copies size entries from T*.

- `~Array ()`
  *Destructor.*

- `Array<T> &operator= (const Array<T> &a)`
  *Normal const assignment operator.*

- `Array<T> &operator= (Array<T> &a)`
  *Normal assignment operator.*

- `Array<T> &operator= (const T &ival)`
  *Sets all elements in self to the value ival.*

- `operator T * () const`
  *Converts the Array to a standard C-style array. Use with care!*

- `T & operator[ ] (int i)`
  *Alternate bounds-checked indexing operator for int indices*

- `const T & operator[ ] (int i) const`
  *Alternate bounds-checked const indexing operator for int indices*
- T & \texttt{operator[] (size_t i)} \\
  \textit{Index operator, returns the ith value of the array.}

- const T & \texttt{operator[] (size_t i) const} \\
  \textit{Index operator const, returns the ith value of the array.}

- T & \texttt{operator() (size_t i)} \\
  \textit{Index operator, not bounds checked.}

- const T & \texttt{operator() (size_t i) const} \\
  \textit{Index operator const, not bounds checked.}

- \texttt{void read (istream \&s)} \\
  \textit{Reads an Array from an istream.}

- \texttt{void write (ostream \&s) const} \\
  \textit{Writes an Array to an output stream.}

- \texttt{void write (ostream \&s, const Array< String > \&label_array) const} \\
  \textit{Writes an Array and associated label array to an output stream.}

- \texttt{void write_aprepro (ostream \&s, const Array< String > \&label_array) const} \\
  \textit{aprepro format}

- \texttt{void write.annotated (ostream \&s, bool write_len) const} \\
  \textit{Writes an Array to an output stream in annotated format.}

- \texttt{void read (BiStream \&s)} \\
  \textit{Reads an Array from a binary input stream.}

- \texttt{void write (BoStream \&s) const} \\
  \textit{Writes an Array to a binary output stream.}

- \texttt{void read (MPIUnpackBuffer \&s)} \\
  \textit{Reads an Array from a buffer after an MPI receive.}

- \texttt{void write (MPPackBuffer \&s) const} \\
  \textit{Writes an Array to a buffer prior to an MPI send.}

- \texttt{size_t length () const} \\
  \textit{Returns size of array.}

- \texttt{void reshape (size_t sz)} \\
  \textit{Resizes array to size sz.}

- \texttt{size_t index (const T \&a) const}
Returns the index of the first array item which matches the object a.

- bool contains (const T &a) const
  Checks if the array contains an object which matches the object a.

- size_t count (const T &a) const
  Returns the number of items in the array matching the object a.

- const T * data () const
  Returns pointer T* to continuous data.

8.9.1 Detailed Description

template<class T> class Dakota::Array<T>

Template class for the Dakota bookkeeping array.
An array class template that provides additional functionality that is specific to Dakota’s needs. The Array class adds additional functionality needed by Dakota to the inherited base array class. The Array class can inherit from either the STL or RW vector classes.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Array (const T * p, size_t size) [inline]

Constructor which copies size entries from T*.
Assigns size values from p into array.

8.9.3 Member Function Documentation

8.9.3.1 Array<T> & operator= (const T & ival) [inline]

Sets all elements in self to the value ival.
Assigns all values of array to the value passed in as ival. For the Rogue Wave case, utilizes base class operator=(ival), while for the ANSI case, uses the STL assign() method.
8.9 Array Class Template Reference

8.9.3.2 operator T * () const [inline]

Converts the Array to a standard C-style array. Use with care!
The operator() returns a C style pointer to the data within the array. Calls the data() method. USE WITH CARE.

8.9.3.3 T & operator[] (size_t i) [inline]

Index operator, returns the ith value of the array.
Index operator; calls the STL method at() which is bounds checked. Mimics the RW vector class. Note: the at() method is not supported by the __GNUC__ STL implementation or by builds omitting exceptions (e.g., SIERRA).

8.9.3.4 const T & operator[] (size_t i) const [inline]

Index operator const, returns the ith value of the array.
A const version of the index operator; calls the STL method at() which is bounds checked. Mimics the RW vector class. Note: the at() method is not supported by the __GNUC__ STL implementation or by builds omitting exceptions (e.g., SIERRA).

8.9.3.5 T & operator() (size_t i) [inline]

Index operator, not bounds checked.
Non bounds check index operator, calls the STL operator[] which is not bounds checked. Needed to mimic the RW vector class

8.9.3.6 const T & operator() (size_t i) const [inline]

Index operator const, not bounds checked.
A const version of the non-bounds check index operator, calls the STL operator[] which is not bounds checked. Needed to mimic the RW vector class

8.9.3.7 const T * data () const [inline]

Returns pointer T* to continuous data.
Returns a C style pointer to the data within the array. USE WITH CARE. Needed to mimic RW vector class, is used in the operator(). Uses the STL front method.
The documentation for this class was generated from the following file:

- DakotaArray.H
8.10  BaseConstructor Struct Reference

Dummy struct for overloading letter-envelope constructors.

Public Member Functions

- **BaseConstructor**(int=0)
  
  *C++ structs can have constructors.*

8.10.1  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:

- global_defs.h
8.11 BaseVector Class Template Reference

Base class for the Dakota::Matrix and Dakota::Vector classes.

Inheritance diagram for BaseVector::

```
BaseVector
  ↓
Matrix    Vector
```

Public Member Functions

- **BaseVector ()**
  
  *Default constructor.*

- **BaseVector (size_t size)**
  
  *Constructor, creates vector of size.*

- **BaseVector (size_t size, const T &initial_val)**
  
  *Constructor, creates vector of size with initial value of initial_val.*

- **~BaseVector ()**
  
  *Destructor.*

- **BaseVector (const BaseVector<T> &a)**
  
  *Copy constructor.*

- **BaseVector<T> & operator= (const BaseVector<T> &a)**
  
  *Normal assignment operator.*

- **BaseVector<T> & operator= (const T &ival)**
  
  *Assigns all values of vector to ival.*

- **T & operator[] (int i)**
  
  *Alternate bounds-checked indexing operator for int indices*

- **const T & operator[] (int i) const**
  
  *Alternate bounds-checked const indexing operator for int indices*

- **T & operator[] (size_t i)**
  
  *Returns the object at index i, (can use as lvalue).*
• const T & operator[ ] (size_t i) const
  
  *Returns the object at index i, const (can’t use as lvalue).*

• T & operator() (size_t i)
  
  *Index operator, not bounds checked.*

• const T & operator() (size_t i) const
  
  *Index operator const, not bounds checked.*

• size_t length () const
  
  *Returns size of vector.*

• void reshape (size_t sz)
  
  *Resizes vector to size sz.*

• const T * data () const
  
  *Returns const pointer to standard C array. Use with care.*

### Protected Member Functions

• T * array () const
  
  *Returns pointer to standard C array. Use with care.*

### 8.11.1 Detailed Description

template<class T> class Dakota::BaseVector<T>

Base class for the Dakota::Matrix and Dakota::Vector classes.

The Dakota::BaseVector class is the base class for the Dakota::Matrix class. It is used to define a common vector interface for both the STL and RW vector classes. If the STL version is based on the valarray class then some basic vector operations such as +, * are available.

### 8.11.2 Constructor & Destructor Documentation

#### 8.11.2.1 BaseVector (size_t size, const T & initial_val) [inline]

Constructor, creates vector of size with initial value of initial_val.

Constructor which takes an initial size and an initial value, allocates an area of initial size and initializes it with input value. Calls base class constructor
8.11.3 Member Function Documentation

8.11.3.1 \[ T \& operator[ ] (size_t i) \ [inline]\]

Returns the object at index i, (can use as lvalue).
Index operator, calls the STL method at() which is bounds checked. Mimics the RW vector class. Note: the at() method is not supported by the __GNUC__ STL implementation or by builds omitting exceptions (e.g., SIERRA).

8.11.3.2 \[ const T \& operator[ ] (size_t i) const \ [inline]\]

Returns the object at index i, const (can’t use as lvalue).
Const versions of the index operator calls the STL method at() which is bounds checked. Mimics the RW vector class. Note: the at() method is not supported by the __GNUC__ STL implementation or by builds omitting exceptions (e.g., SIERRA).

8.11.3.3 \[ T \& operator() (size_t i) \ [inline]\]

Index operator, not bounds checked.
Non bounds check index operator, calls the STL operator[] which is not bounds checked. Needed to mimic the RW vector class

8.11.3.4 \[ const T \& operator() (size_t i) const \ [inline]\]

Index operator const, not bounds checked.
Const version of the non-bounds check index operator, calls the STL operator[] which is not bounds checked. Needed to mimic the RW vector class

8.11.3.5 \[ size_t length () const \ [inline]\]

Returns size of vector.
Returns the length of the array by calling the STL size method. Needed to mimic the RW vector class

8.11.3.6 \[ void reshape (size_t sz) \ [inline]\]

Resizes vector to size sz.
Resizes the array to size sz by calling the STL resize method. Needed to mimic the RW vector class
8.11.3.7 const T * data () const [inline]

Returns const pointer to standard C array. Use with care.
Returns a const pointer to the data within the array. USE WITH CARE. Needed to mimic RW vector class.

8.11.3.8 T * array () const [inline, protected]

Returns pointer to standard C array. Use with care.
Returns a non-const pointer to the data within the array. Non-const version of data() used by derived classes.
The documentation for this class was generated from the following file:

- DakotaBaseVector.H
8.12 BiStream Class Reference

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(const char *s, int mode)**
  
  *Constructor takes name of input file, mode.*

- **~BiStream()**
  
  *Destructor, calls xdr_destroy to delete xdr stream.*

- **BiStream & operator>>(String &ds)**
  
  *Binary Input stream operator*>

- **BiStream & operator>>(char *s)**
  
  *Input operator, reads char* from binary stream BiStream.*

- **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.

• 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.

8.12.1 Detailed Description

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.

8.12.2 Constructor & Destructor Documentation
8.12.2.1 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.

8.12.2.2 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.

8.12.2.3 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.

8.12.2.4 ~BiStream ()

Destructor, calls xdr_destroy to delete xdr stream.
Destructor, destroys the xdr stream allocated in constructor

8.12.3 Member Function Documentation

8.12.3.1 BiStream & operator >> (String & ds)

Binary Input stream operator >>.
The String input operator must first read both the xdr buffer size and the size of the string written. Once these our read it can then read and convert the String correctly.

8.12.3.2 BiStream & operator >> (char * s)

Input operator, reads char* from binary stream BiStream.
Reading char array is a special case. The method has no way of knowing if the length to the input array is large enough, it assumes it is one char longer than actual string, (Null terminator added). As with the String the size of the xdr buffer as well as the char array size written must be read from the stream prior to reading and converting the char array.
The documentation for this class was generated from the following files:
- DakotaBinStream.H
- DakotaBinStream.C
8.13 BoStream Class Reference

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.

- **BoStream (const char *s, int mode)**
  
  Constructor takes name of input file, mode.

- **~BoStream ()**
  
  Destructor, calls xdr_destroy to delete xdr stream.

- **BoStream & operator<< (const String &ds)**
  
  Binary Output stream operator<<.

- **BoStream & operator<< (const char *s)**
  
  Output operator, writes char* TO binary stream BoStream.

- **BoStream & operator<< (const char &c)**
  
  Output operator, writes char to binary stream BoStream.

- **BoStream & operator<< (const int &i)**
  
  Output operator, writes int to binary stream BoStream.

- **BoStream & operator<< (const long &l)**
  
  Output operator, writes long to binary stream BoStream.

- **BoStream & operator<< (const short &s)**
  
  Output operator, writes short to binary stream BoStream.

- **BoStream & operator<< (const bool &b)**
  
  Output operator, writes bool to binary stream BoStream.
• BoStream & operator<<(const double &d)
  Output operator, writes double to binary stream BoStream.

• BoStream & operator<<(const float &f)
  Output operator, writes float to binary stream BoStream.

• BoStream & operator<<(const unsigned char &c)
  Output operator, writes unsigned char to binary stream BoStream.

• BoStream & operator<<(const unsigned int &i)
  Output operator, writes unsigned int to binary stream BoStream.

• BoStream & operator<<(const unsigned long &l)
  Output operator, writes unsigned long to binary stream BoStream.

• BoStream & operator<<(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.

8.13.1 Detailed Description

data types

The Dakota::BoStream class is a binary output classes which overloads the << 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.

8.13.2 Constructor & Destructor Documentation
**8.13 BoStream Class Reference**

### 8.13.2.1 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.

### 8.13.2.2 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.

### 8.13.2.3 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.

### 8.13.3 Member Function Documentation

#### 8.13.3.1 BoStream & operator<< (const String & ds)

Binary Output stream operator<<.

The 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 String.

#### 8.13.3.2 BoStream & operator<< (const char * s)

Output operator, writes char* TO binary stream BoStream.

The output of char* is the same as the output of the String. The size of the xdr buffer and the size of the string must be written first, then the string itself.

The documentation for this class was generated from the following files:

- DakotaBinStream.H
- DakotaBinStream.C
8.14 COLINApplication Class Reference

Public Member Functions

- **COLINApplication (Model &model)**
  constructor

- **~COLINApplication ()**
  destructor

- **void DoEval (ColinPoint &point, int &priority, ColinResponse &response, bool synch_flag)**
  launch a function evaluation either synchronously or asynchronously

- **unsigned int num_evaluation_servers ()**
  The value '0' indicates that this is a sequential application.

- **void synchronize ()**
  blocking retrieval of all pending jobs

- **int next_eval ()**
  nonblocking query and retrieval of a job if completed

- **void blocking_synch (const bool &blocking_synch)**
  construct time.

- **void dakota_asynch_flag (const bool &asynch_flag)**
  (asynchFlag not initialized properly at construction).

Private Member Functions

- **void map_response (ColinResponse &colin_response, const Response &dakota_response)**
  utility function for mapping a DAKOTA response to a COLIN response

Private Attributes

- **Model & iteratedModel**
  reference to the COLINOptimizer’s model passed in the constructor

- **ActiveSet activeSet**
  copy/conversion of the COLIN request vector
8.14 COLINApplication Class Reference

- bool dakotaModelAsynchFlag
  a flag for asynchronous DAKOTA evaluations

- bool blockingSynch
  needed for APPS, to enforce blocking synch despite call of next_eval().

- IntResponseMap dakotaResponseMap
  map of DAKOTA responses returned by synchronize_nowait()

- size_t numObjFns
  number of objective functions

- size_t numNonlinCons
  number of nonlinear constraints

- int num_real_params
  number of continuous design variables

- int num_integer_params
  number of discrete design variables

- int synchronization_state
  tracks the state of asynchronous evaluations

8.14.1 Detailed Description

COLINApplication is a DAKOTA class that is derived from COLIN’s OptApplication 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.

8.14.2 Member Function Documentation

8.14.2.1 void DoEval (ColinPoint & pt, int & priority, ColinResponse * prob_response, bool synch_flag)

launch a function evaluation either synchronously or asynchronously

Converts the ColinPoint variables and request vector to DAKOTA variables and active set vector, performs a DAKOTA function evaluation with synchronization governed by synch_flag, and then copies the Response data to the ColinResponse response (synchronous) or bookkeeps the response object (asynchronous).
8.14.2.2 void synchronize ()

blocking retrieval of all pending jobs
Blocking synchronize of asynchronous DAKOTA jobs followed by conversion of the Response objects to Colin-Response response objects.

8.14.2.3 int next_eval ()

nonblocking query and retrieval of a job if completed
Nonblocking job retrieval. Finds a completion (if available), populates the COLIN response, and sets id to the completed job's id. Else set id = -1.

8.14.2.4 void map_response (ColinResponse & colin_response, const Response & dakota_response)

utility function for mapping a DAKOTA response to a COLIN response
map_response Maps a Response object into a ColinResponse class that is compatible with COLIN.
The documentation for this class was generated from the following files:

- COLINApplication.H
- COLINApplication.C
8.15 COLINOptimizer Class Template Reference

Wrapper class for optimizers defined using COLIN.

Inheritance diagram for COLINOptimizer::

![Inheritance Diagram]

Public Member Functions

- **COLINOptimizer (Model &model)**
  
- **COLINOptimizer (Model &model, int seed)**
  alternate constructor for on-the-fly instantiations

- **~COLINOptimizer ()**
  destructor

- **void find_optimum ()**
  Performs the iterations to determine the optimal solution.

- **bool returns_multiple_points () const**
  COLINY methods can return multiple points.

- **template<> void set_method_parameters ()**
- **template<> void set_method_parameters ()**
- **template<> void set_method_parameters ()**
- **template<> void set_runtime_parameters ()**
- **template<> void set_method_parameters ()**
- **template<> void get_final_points ()**
- **template<> void get_final_points ()**
- **template<> void get_final_points ()**
Protected Member Functions

- virtual void set_rng (int seed)
  sets up the random number generator for stochastic methods

- virtual void set_initial_point (ColinPoint &pt)
  sets the iteration starting point prior to minimization

- virtual void get_min_point (ColinPoint &pt)
  retrieves the final solution after minimization

- virtual void set_method_parameters ()
  (called at construction time)

- void set_standard_method_parameters ()
  sets the standard method parameters shared by all methods

- virtual void set_runtime_parameters ()
  not available until run time

- virtual void get_final_points ()
  Get the set of best points from the solver.

- void resize_final_points (size_t newsize)
  resize bestVariablesArray

Protected Attributes

- OptimizerT * optimizer
  Pointer to COLIN base optimizer object.

- COLINApplication * application
  Pointer to the COLINApplication object.

- colin::OptProblem< ColinPoint > problem
  the COLIN problem object

- utilib::RNG * rng
  RNG ptr.

- bool blockingSynch
  nonblocking

- Real solverStartTime
  Start time for keeping track of time for solver to run.
8.15 COLINOOptimizer Class Template Reference

- Real solverTime
  
  Time taken by solver to run.

8.15.1 Detailed Description

template<class OptimizerT> class Dakota::COLINOOptimizer< OptimizerT >

Wrapper class for optimizers defined using COLIN.

The COLINOOptimizer class provides a templated wrapper for COLIN, a Sandia-developed C++ optimization interface library. A variety of COLIN optimizers are defined in the COLINY optimization library, which contains the optimization components from the old SGOPT library. COLINY contains optimizers such as genetic algorithms, pattern search methods, and other nongradient-based techniques. COLINOOptimizer uses a COLINApplication object to perform the function evaluations.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, solution_accuracy and max_cpu_time are mapped into COLIN’s max_iters, max_neval, ftol, accuracy, and max_time data attributes. An output setting of verbose is passed to COLIN’s set_output() function and a setting of debug activates output of method initialization and sets the COLIN debug attribute to 10000. Refer to [Hart, W.E., 2006] for additional information on COLIN objects and controls.

8.15.2 Member Function Documentation

8.15.2.1 void find_optimum () [inline, virtual]

Performs the iterations 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 minimize() on the COLIN optimizer, and finally catalogues the results.

Implements Optimizer.

8.15.2.2 void set_standard_method_parameters () [inline, protected]

sets the standard method parameters shared by all methods

set_standard_method_parameters propagates standard DAKOTA user input to the optimizer.

8.15.2.3 void set_method_parameters () [inline]

specialization of set_method_parameters() for DIRECT
8.15.2.4 void set_method_parameters () [inline]
specialization of set_method_parameters() for Cobyla

8.15.2.5 void set_method_parameters () [inline]
specialization of set_method_parameters() for APPS

8.15.2.6 void set_runtime_parameters () [inline]
specialization of set_runtime_parameters() for PatternSearch

8.15.2.7 void set_method_parameters () [inline]
specialization of set_method_parameters() for PatternSearch

8.15.2.8 void set_method_parameters () [inline]
specialization of set_method_parameters() for SolisWets

8.15.2.9 void set_method_parameters () [inline]
specialization of set_method_parameters() for EAmnlp

The documentation for this class was generated from the following file:

- COLINOptimizer.H
8.16 ColinPoint Class Reference

Public Attributes

- vector< double > rvec
  
  *continuous parameter values*

- vector< int > ivec
  
  *discrete parameter values*

8.16.1 Detailed Description

A class containing a vector of doubles and integers.

The documentation for this class was generated from the following file:

- COLINApplication.H
8.17 CommandLineHandler Class Reference

Utility class for managing command line inputs to DAKOTA.

Inheritance diagram for CommandLineHandler::

```
  GetLongOpt
   ...
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)**
  
  *Prints a descriptive message and exits the program if incorrect.*

- **int read_restart_evals () const**
  
  *instead of a const char*.

Private Member Functions

- **void initialize_options ()**
  
  *enrolls the supported command line inputs.*

- **void output_version (ostream &s) const**
  
  *outputs the DAKOTA version*
8.17 CommandLineHandler Class Reference

8.17.1 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.

The documentation for this class was generated from the following files:

- CommandLineHandler.H
- CommandLineHandler.C
8.18 CommandShell Class Reference

processes with system calls.

Public Member Functions

- **CommandShell ()**
  
  *constructor*

- **~CommandShell ()**
  
  *destructor*

- **CommandShell & operator<< (const char *string)**
  
  *adds string to unixCommand*

- **CommandShell & operator<< (CommandShell &+(+[](CommandShell &))**
  
  *allows passing of the flush function to the shell using <<*

- **CommandShell & flush ()**
  
  *“flushes” the shell; i.e. executes the unixCommand*

- **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

- **String unixCommand**
  
  *insertions and then executed by flush*

- **bool asynchFlag**
  
  *flags nonblocking operation (background system calls)*
8.18 CommandShell Class Reference

- bool suppressOutputFlag
  flags suppression of shell output (no command echo)

8.18.1 Detailed Description

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.

8.18.2 Member Function Documentation

8.18.2.1 CommandShell & flush ()

"flushes" the shell; i.e. executes the unixCommand

Executes the unixCommand by passing it to system(). Appends an "&" if asynchFlag is set (background system call) and echos the unixCommand to Cout if suppressOutputFlag is not set.
The documentation for this class was generated from the following files:

- CommandShell.H
- CommandShell.C
8.19 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

- void **run_strategy** ()

  settings within the iterator or model.

Private Member Functions

- void **self_schedule_iterators** ()

  among slave iterator servers (called by run_strategy())

- void **serve_iterators** ()

  assigned by the strategy master (called by run_strategy())

- void **static_schedule_iterators** ()

  (called by run_strategy())

- void **print_results** ()

  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

- **int numIteratorServers**
  number of concurrent iterator partitions

- **int numIteratorJobs**
  total number of iterator executions to schedule over the servers

- **RealVectorArray parameterSets**
  sets or pareto multiobjective weighting sets) to be performed.

- **PRPArray prpResults**
  an array of results corresponding to the parameter set vectors.

- **bool multiStartFlag**
  distinguishes multi-start from Pareto-set

- **bool strategyDedicatedMasterFlag**
  signals ded. master partitioning

- **int iteratorServerId**
  identifier for an iterator server

- **int drvMsgLen**
  length of an MPI buffer containing a RealVector from parameterSets

### 8.19.1 Detailed Description

**Strategy** for multi-start iteration or pareto set optimization.

This strategy maintains two concurrent iterator capabilities. First, a general capability for running an iterator multiple times from different starting points is provided (often used for multi-start optimization, but not restricted to optimization). Second, a simple capability for mapping the "pareto frontier" (the set of optimal solutions in multiobjective formulations) is provided. This pareto set is mapped through running an optimizer multiple times for different sets of multiobjective weightings.

### 8.19.2 Member Function Documentation
8.19.2.1  void self_schedule_iterators () [private]

among slave iterator servers (called by run_strategy())
This function is adapted from ApplicationInterface::self_schedule_evaluations().

8.19.2.2  void serve_iterators () [private]

assigned by the strategy master (called by run_strategy())
This function is similar in structure to ApplicationInterface::serve_evaluations_synch().

The documentation for this class was generated from the following files:

- ConcurrentStrategy.H
- ConcurrentStrategy.C
8.20 CONMINOptimizer Class Reference

Wrapper class for the CONMIN optimization library.

Inheritance diagram for CONMINOptimizer::

```
  Iterator
   ↓
  Minimizer
   ↓
  Optimizer
   ↓
CONMINOptimizer
```

Public Member Functions

- **CONMINOptimizer (Model &model)**
  
  *constructor*

- **~CONMINOptimizer ()**
  
  *destructor*

- **void find_optimum ()**
  
  *Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- **virtual void derived_pre_run ()**
  
  *performs run-time set up*

Private Member Functions

- **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

• SizetList constraintMappingIndices
  Response constraints used in computing the CONMIN constraints.

• RealList constraintMappingMultipliers
  the CONMIN constraints.

• RealList constraintMappingOffsets
  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 CDMIN
  
  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 N1 = numdv+2).

- double * conminLowerBnds
Array of lower bounds used by CONMIN (length N1 = numdv+2).

- double * conminUpperBnds
  Array of upper bounds used by CONMIN (length N1 = 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.

### 8.20.1 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 CONMIN parameter equivalent, convergence_tolerance is mapped into CONMIN’s DELFUN and DABFUN parameters, output verbosity is mapped into CONMIN’s IPRINT parameter (verbose: IPRINT = 4; quiet: IPRINT = 2), gradient mode is mapped into CONMIN’s 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.

8.20.2 Member Data Documentation

8.20.2.1 int conminInfo [private]
INFO from CONMIN manual.
Information requested by CONMIN: 1 = evaluate objective and constraints, 2 = evaluate gradients of objective and constraints.

8.20.2.2 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

8.20.2.3 int optimizationType [private]
MINMAX from DOT manual (minimize or maximize).
Values of 0 or -1 (minimize) or 1 (maximize).

8.20.2.4 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 \( \leq 0 \) (which requires a transformation from 2-sided inequalities and equalities).

8.20.2.5 SizetList constraintMappingIndices [private]
Response constraints used in computing the CONMIN constraints.
The length of the list corresponds to the number of CONMIN constraints, and each entry in the list points to the corresponding DAKOTA constraint.

### 8.20.2.6 RealList `constraintMappingMultipliers` [private]

The length of the list corresponds to the number of CONMIN constraints, and each entry in the list contains a multiplier for the DAKOTA constraint identified with `constraintMappingIndices`. These multipliers are currently +1 or -1.

### 8.20.2.7 RealList `constraintMappingOffsets` [private]

CONMIN constraints. The length of the list corresponds to the number of CONMIN constraints, and each entry in the list contains an offset for the DAKOTA constraint identified with `constraintMappingIndices`. These offsets involve inequality bounds or equality targets, since CONMIN assumes constraint allowables = 0.

### 8.20.2.8 int `N1` [private]

Size variable for CONMIN arrays. See CONMIN manual.

\[ N1 = \text{number of variables} + 2 \]

### 8.20.2.9 int `N2` [private]

Size variable for CONMIN arrays. See CONMIN manual.

\[ N2 = \text{number of constraints} + 2 \times (\text{number of variables}) \]

### 8.20.2.10 int `N3` [private]

Size variable for CONMIN arrays. See CONMIN manual.

\[ N3 = \text{Maximum possible number of active constraints.} \]

### 8.20.2.11 int `N4` [private]

Size variable for CONMIN arrays. See CONMIN manual.

\[ N4 = \text{Maximum}(N3, \text{number of variables}) \]

### 8.20.2.12 int `N5` [private]

Size variable for CONMIN arrays. See CONMIN manual.

\[ N5 = 2 \times (N4) \]
8.20.2.13 double CT [private]
Constraint thickness parameter.
The value of CT decreases in magnitude during optimization.

8.20.2.14 double* S [private]
Internal CONMIN array.
Move direction in N-dimensional space.

8.20.2.15 double* G1 [private]
Internal CONMIN array.
Temporary storage of constraint values.

8.20.2.16 double* G2 [private]
Internal CONMIN array.
Temporary storage of constraint values.

8.20.2.17 double* B [private]
Internal CONMIN array.
Temporary storage for computations involving array S.

8.20.2.18 double* C [private]
Internal CONMIN array.
Temporary storage for use with arrays B and S.

8.20.2.19 int* MS1 [private]
Internal CONMIN array.
Temporary storage for use with arrays B and S.

8.20.2.20 double* SCAL [private]
Internal CONMIN array.
Vector of scaling parameters for design parameter values.
8.20.2.21 double* DF [private]

Internal CONMIN array.
Temporary storage for analytic gradient data.

8.20.2.22 double* A [private]

Internal CONMIN array.
Temporary 2-D array for storage of constraint gradients.

8.20.2.23 int* ISC [private]

Internal CONMIN array.
Array of flags to identify linear constraints. (not used in this implementation of CONMIN)

8.20.2.24 int* IC [private]

Internal CONMIN array.
Array of flags to identify active and violated constraints

The documentation for this class was generated from the following files:

- CONMINOptimizer.H
- CONMINOptimizer.C
8.21 Constraints Class Reference

Base class for the variable constraints class hierarchy.

Inheritance diagram for Constraints::

```
Constraints
  |       |
  |       |
AllConstraints DistinctConstraints MergedConstraints
```

Public Member Functions

- **Constraints ()**
  *default constructor*
- **Constraints (const ProblemDescDB &problem_db, const pair< short, short >&view)**
  *standard constructor*
- **Constraints (const pair< short, short >&view)**
  *alternate constructor for instantiations on the fly*
- **Constraints (const Constraints &con)**
  *copy constructor*
- **virtual ~Constraints ()**
  *destructor*
- **Constraints operator= (const Constraints &con)**
  *assignment operator*
- **virtual const RealVector & continuous_lower_bounds () const**
  *return the active continuous variable lower bounds*
- **virtual void continuous_lower_bounds (const RealVector &c_l_bnds)**
  *set the active continuous variable lower bounds*
- **virtual const RealVector & continuous_upper_bounds () const**
  *return the active continuous variable upper bounds*
- **virtual void continuous_upper_bounds (const RealVector &c_u_bnds)**
  *set the active continuous variable upper bounds*
• virtual const IntVector & discrete_lower_bounds () const
  return the active discrete variable lower bounds

• virtual void discrete_lower_bounds (const IntVector &d_l_bnds)
  set the active discrete variable lower bounds

• virtual const IntVector & discrete_upper_bounds () const
  return the active discrete variable upper bounds

• virtual void discrete_upper_bounds (const IntVector &d_u_bnds)
  set the active discrete variable upper bounds

• virtual const RealVector & inactive_continuous_lower_bounds () const
  return the inactive continuous lower bounds

• virtual void inactive_continuous_lower_bounds (const RealVector &i_c_l_bnds)
  set the inactive continuous lower bounds

• virtual const RealVector & inactive_continuous_upper_bounds () const
  return the inactive continuous upper bounds

• virtual void inactive_continuous_upper_bounds (const RealVector &i_c_u_bnds)
  set the inactive continuous upper bounds

• virtual const IntVector & inactive_discrete_lower_bounds () const
  return the inactive discrete lower bounds

• virtual void inactive_discrete_lower_bounds (const IntVector &i_d_l_bnds)
  set the inactive discrete lower bounds

• virtual const IntVector & inactive_discrete_upper_bounds () const
  return the inactive discrete upper bounds

• virtual void inactive_discrete_upper_bounds (const IntVector &i_d_u_bnds)
  set the inactive discrete upper bounds

• virtual RealVector all_continuous_lower_bounds () const
  returns a single array with all continuous lower bounds

• virtual void all_continuous_lower_bounds (const RealVector &a_c_l_bnds)
  sets all continuous lower bounds using a single array

• virtual RealVector all_continuous_upper_bounds () const
  returns a single array with all continuous upper bounds
virtual void all_continuous_upper_bounds(const RealVector &a_c_u_bnds) 
sets all continuous upper bounds using a single array

virtual IntVector all_discrete_lower_bounds() const 
returns a single array with all discrete lower bounds

virtual void all_discrete_lower_bounds(const IntVector &a_d_l_bnds) 
sets all discrete lower bounds using a single array

virtual IntVector all_discrete_upper_bounds() const 
returns a single array with all discrete upper bounds

virtual void all_discrete_upper_bounds(const IntVector &a_d_u_bnds) 
sets all discrete upper bounds using a single array

virtual void write (ostream &s) const 
write a variable constraints object to an ostream

virtual void read (istream &s) 
read a variable constraints object from an istream

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
void linear_eq_constraint_coeffs (const RealMatrix &lin_eq_coeffs)
set the linear equality constraint coefficients

const RealVector & linear_eq_constraint_targets () const
return the linear equality constraint targets

void linear_eq_constraint_targets (const RealVector &lin_eq_targets)
set the linear equality constraint targets

size_t num_nonlinear_ineq_constraints () const
return the number of nonlinear inequality constraints

size_t num_nonlinear_eq_constraints () const
return the number of nonlinear equality constraints

const RealVector & nonlinear_ineq_constraint_lower_bounds () const
return the nonlinear inequality constraint lower bounds

void nonlinear_ineq_constraint_lower_bounds (const RealVector &nln_ineq_l_bnds)
set the nonlinear inequality constraint lower bounds

const RealVector & nonlinear_ineq_constraint_upper_bounds () const
return the nonlinear inequality constraint upper bounds

void nonlinear_ineq_constraint_upper_bounds (const RealVector &nln_ineq_u_bnds)
set the nonlinear inequality constraint upper bounds

const RealVector & nonlinear_eq_constraint_targets () const
return the nonlinear equality constraint targets

void nonlinear_eq_constraint_targets (const RealVector &nln_eq_targets)
set the nonlinear equality constraint targets

Constraints copy () const
for use when a deep copy is needed (the representation is not shared)

void reshape (const size_t &num_nln_ineq_cons, const size_t &num_nln_eq_cons, const size_t &num_lin_ineq_cons, const size_t &num_lin_eq_cons)
Constraints hierarchy.

void reshape (const Sizet2DArray &vars_comps)
reshape the bounds arrays within the Constraints hierarchy

bool is_null () const
function to check constraintsRep (does this envelope contain a letter)
Protected Member Functions

- **Constraints** (BaseConstructor, const ProblemDescDB &problem_db, const pair< short, short >&view)
  
  *derived class constructors - Coplien, p. 139*)

- virtual void copy_rep (const Constraints *con_rep)
  
  *Used by copy() to copy the contents of a letter class.*

- virtual void reshape_rep (const Sizet2DArray &vars_comps)
  
  *Used by reshape(Sizet2DArray&) to rehape the contents of a letter class.*

- void manage_linear_constraints (const ProblemDescDB &problem_db)
  
  *coefficient input to matrices, and assign defaults*

Protected Attributes

- pair< short, short > variablesView
  
  *view enumerations*

- 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 emptyRealVector`  
  no variable constraints corresponding to the request

- `IntVector emptyIntVector`  
  no variable constraints corresponding to the request

**Private Member Functions**

- `Constraints * get_constraints (const ProblemDescDB &problem_db, const pair< short, short > &view)`  
  appropriate derived type.

- `Constraints * get_constraints (const pair< short, short > &view) const`  
  derived type.

**Private Attributes**

- `Constraints * constraintsRep`  
  pointer to the letter (initialized only for the envelope)

- `int referenceCount`  
  number of objects sharing constraintsRep

### 8.21.1 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.
8.21 Constructor & Destructor Documentation

8.21.1 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.

8.21.2 Constraints (const ProblemDescDB & problem_db, const pair< short, short > & view)

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.

8.21.3 Constraints (const pair< short, short > & view)

alternate constructor for instantiations on the fly

Envelope constructor for instantiations on the fly.

8.21.4 Constraints (const Constraints & con)

copy constructor

Copy constructor manages sharing of constraintsRep and incrementing of referenceCount.

8.21.5 ~Constraints () [virtual]

destructor

Destructor decrements referenceCount and only deletes constraintsRep when referenceCount reaches zero.

8.21.6 Constraints (BaseConstructor, const ProblemDescDB & problem_db, const pair< short, short > & view) [protected]

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).
8.21.3  Member Function Documentation

8.21.3.1  Constraints operator=(const Constraints & con)

assignment operator
Assignment operator decrements referenceCount for old constraintsRep, assigns new constraintsRep, and incre-
ments referenceCount for new constraintsRep.

8.21.3.2  Constraints copy () const

for use when a deep copy is needed (the representation is _not_ shared)
Deep copies are used for history mechanisms such as bestVariables and data_pairs since these must catalogue
copies (and should not change as the representation within currentVariables changes).

8.21.3.3  void reshape (const size_t & num_nln_ineq_cons, const size_t & num_nln_eq_cons, const size_t &
num_lin_ineq_cons, const size_t & num_lin_eq_cons)

Constraints hierarchy.
Resizes the linear and nonlinear constraint arrays at the base class. Does NOT currently resize the derived bounds
arrays.

8.21.3.4  void reshape (const Sizet2DArray & vars_comps)

reshape the bounds arrays within the Constraints hierarchy
Resizes the derived bounds arrays.

8.21.3.5  void manage_linear_constraints (const ProblemDescDB & problem_db)  [protected]

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.

8.21.3.6  Constraints * get_constraints (const ProblemDescDB & problem_db, const pair< short, short >
& view)  [private]

appropriate derived type.
Initializes constraintsRep to the appropriate derived type, as given by the variables view.
8.21.3.7 **Constraints** = get_constraints (const pair< short, short > & view) const [private]

derived type.

Initializes constraintsRep to the appropriate derived type, as given by the variables view. The default derived class constructors are invoked.

The documentation for this class was generated from the following files:

- DakotaConstraints.H
- DakotaConstraints.C
8.22 CtelRegexp Class Reference

Public Types

- enum RStatus { GOOD = 0, EXP_TOO_BIG, OUT_OF_MEM, TOO_MANY_PAR, UNMATCH_PAR, STARPLUS_EMPTY, STARPLUS_NESTED, INDEX_RANGE, INDEX_MATCH, STARPLUS NOTHING, TRAILING, INT_ERROR, BAD_PARAM, BAD_OPCODE }

occurs with this implementation.

Public Member Functions

- CtelRegexp (const std::string &pattern)
  Constructor - compile a regular expression.

- ~CtelRegexp ()
  Destructor.

- bool compile (const std::string &pattern)
  Compile a new regular expression.

- bool match (const std::string &str, size_t start, size_t size)
  another form of matching; returns the indexes of the matching

- RStatus getStatus ()
  Get status.

- const std::string & getStatusMsg ()
  Get status message.

- void clearErrors ()
  Clear all errors.

- const std::string & getRe ()
  Return regular expression pattern.

- bool split (const std::string &str, std::vector< std::string > &all_matches)
**Private Member Functions**

- **CtelRegexp** (const CtelRegexp &)
  *Private copy constructor.*

- **CtelRegexp & operator=** (const CtelRegexp &)
  *Private assignment operator.*

**Private Attributes**

- std::string **strPattern**
  *STL string to hold pattern.*

- regexp * **r**
  *Pointer to regexp.*

- **RStatus status**
  *Return status, enumerated type.*

- std::string **statusMsg**
  *STL string to hold status message.*

**8.22.1 Detailed Description**

**DESCRIPTION:** Wrapper for the Regular Expression engine( regexp ) released by Henry Spencer of the University of Toronto.

The documentation for this class was generated from the following files:

- CtelRegExp.H
- CtelRegExp.C
8.23 DataFitSurrModel Class Reference

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 pair< short, short > &view, const ActiveSet &set, const String &approx_type, const short &approx_order, const String &corr_type, const short &corr_order, const String &sample_reuse)**
  
  alternate constructor for instantiations on the fly

- **~DataFitSurrModel ()**
  
  destructor

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 ResponseArray & 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 **surrogate_bypass** (bool bypass_flag)
  
  any lower-level surrogates.

- void **surrogate_function_indices** (const IntSet &surr_fn_indices)
  
  and ApproximationInterface::approxFnIndices

- void **build_approximation** ()
  
  daceIterator/actualModel to generate new data points

- bool **build_approximation** (const Variables &vars, const Response &response)
  
  augment the vars/response anchor point

- void **update_approximation** (const Variables &vars, const Response &response, bool rebuild_flag)
  
  approximation if requested

- void **update_approximation** (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  
  approximation if requested

- void **append_approximation** (const Variables &vars, const Response &response, bool rebuild_flag)
  
  requested (requests forwarded to approxInterface)

- void **append_approximation** (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  
  rebuilds it if requested (requests forwarded to approxInterface)

- **Array< Approximation >** & **approximations** ()
  
  retrieve the set of Approximations from approxInterface
const RealVectorArray & approximation_coefficients ()
(request forwarded to approxInterface)

void approximation_coefficients (const RealVectorArray &approx_coeffs)
(request forwarded to approxInterface)

void print_coefficients (ostream &s, size_t index) const
(request forwarded to approxInterface)

const RealVector & approximation_variances (const RealVector &c_vars)
(request forwarded to approxInterface)

const List< SurrogateDataPoint > & approximation_data (size_t index)
(request forwarded to approxInterface)

void component_parallel_mode (short mode)
update component parallel mode for supporting parallelism in actualModel

void derived_init_communicators (const 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 (const int &max_iterator_concurrency, bool recurse_flag=true)
set active parallel configuration within actualModel

void derived_free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
(request forwarded to actualModel)

void serve ()
Completes when a termination message is received from stop_servers().

void stop_servers ()
when DataFitSurrModel iteration is complete.

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 ()
(request forwarded to approxInterface and actualModel)
void print_evaluation_summary (ostream &s, bool minimal_header=false, bool relative_count=true) const

(request forwarded to approxInterface and actualModel)

Private Member Functions

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

Private Attributes

int surrModelEvals
derived_asynch_compute_response()

String sampleReuse
all, region, file, or none (default)

String sampleReuseFile
file name for sampleReuse == "file"

Interface approxInterface
(required for both global and local)

Model actualModel
(optional for global, required for local)

Iterator daceIterator
(optional for global since restart data may also be used)
8.23.1 Detailed Description

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.

8.23.2 Member Function Documentation

8.23.2.1 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.

8.23.2.2 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.

8.23.2.3 const ResponseArray & 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.

8.23.2.4 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.

8.23.2.5 void build_approximation () [protected, virtual]
daceIterator/actualModel to generate new data points
This function constructs a new approximation, discarding any previous data. It constructs any required currentPoints and does not define an anchorPoint.
Reimplemented from Model.

8.23.2.6 bool build_approximation (const Variables & vars, const Response & response) [protected, virtual]
augment the vars/response anchor point
This function constructs a new approximation, discarding any previous data. It uses the passed data to populate the anchorPoint and constructs any required currentPoints.
Reimplemented from Model.

8.23.2.7 void update_approximation (const Variables & vars, const Response & response, bool rebuild_flag) [protected, virtual]
approximation if requested
This function populates/replaces Approximation::anchorPoint and rebuilds the approximation, if requested. It does not clear other data (i.e., Approximation::currentPoints) 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.

8.23.2.8 void update_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array, bool rebuild_flag) [protected, virtual]
approximation if requested
This function populates/replaces Approximation::currentPoints and rebuilds the approximation, if requested. It does not clear other data (i.e., Approximation::anchorPoint) 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.
8.23.2.9  void append_approximation (const Variables & vars, const Response & response, bool rebuild_flag)  [protected, virtual]

requested (requests forwarded to approxInterface)
This function appends one point to Approximation::currentPoints and rebuilds the approximation, if requested. It does not modify other data (i.e., Approximation::anchorPoint) 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.

8.23.2.10  void append_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array, bool rebuild_flag)  [protected, virtual]

rebuilds it if requested (requests forwarded to approxInterface)
This function appends multiple points to Approximation::currentPoints and rebuilds the approximation, if requested. It does not modify other data (i.e., Approximation::anchorPoint) 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.

8.23.2.11  void derived_init_communicators (const 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 actualModel will see, and recomputing the message_lengths on the sub-model is probably not a bad idea either. Therefore, recompute everything on actualModel using init_communicators.
Reimplemented from Model.

8.23.2.12  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.

8.23.2.13  void build_global ()  [private]

Builds a global approximation using daceIterator.
Determine sample 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).

### 8.23.2.14 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.

### 8.23.2.15 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.

### 8.23.2.16 void update_from_actual_model () [private]

update current variables/labels/bounds/targets with data from actualModel

Update values and labels in currentVariables and bound/linear/nonlinear constraints in userDefinedConstraints from variables and constraints data within actualModel.

### 8.23.3 Member Data Documentation

#### 8.23.3.1 Model actualModel [private]

(optional for global, required for local)

actualModel is unrestricted in type; arbitrary nestings are possible.

The documentation for this class was generated from the following files:

- DataFitSurrModel.H
- DataFitSurrModel.C
8.24 **DataInterface Class Reference**

Container class for interface specification data.

**Public Member Functions**

- **DataInterface ()**
  
  *constructor*

- **DataInterface (const DataInterface &)**
  
  *copy constructor*

- **~DataInterface ()**
  
  *destructor*

- **DataInterface & operator= (const DataInterface &)**
  
  *assignment operator*

- **bool operator== (const DataInterface &)**
  
  *equality operator*

- **void write (ostream &s) const**
  
  *write a DataInterface object to an 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*

**Public Attributes**

- **String idInterface**
  
  *(from the id_interface specification in InterfIndControl)*

- **String interfaceType**
  
  *the interface selection: system, fork, direct, or grid*

- **String algebraicMappings**
  
  *(AMPL format) from JAGUAR.*
- **StringArray analysisDrivers**
  (from the analysis_drivers specification in InterfIndControl)

- **String2DArray analysisComponents**
  (from the analysis_components specification in InterfIndControl)

- **String inputFilter**
  input_filter specification in InterfIndControl)

- **String outputFilter**
  output_filter specification in InterfIndControl)

- **String parametersFile**
  InterfApplicF)

- **String resultsFile**
  InterfApplicF)

- **String analysisUsage**
  (from the analysis_usage specification in InterfApplicSC)

- **bool apreproFormatFlag**
  specification in InterfApplicSC and InterfApplicF)

- **bool fileTagFlag**
  specification in InterfApplicSC and InterfApplicF)

- **bool fileSaveFlag**
  specification in InterfApplicSC and InterfApplicF)

- **int procsPerAnalysis**
  processors_per_analysis specification in InterfApplicDF)

- **StringArray gridHostNames**
  hostnames specification in InterfApplicG)

- **IntArray gridProcsPerHost**
  processors_per_host specification in InterfApplicG)

- **String interfaceSynchronization**
  InterfIndControl)

- **int asynchLocalEvalConcurrency**
  the evaluation_concurrency specification in InterfIndControl)

- **int asynchLocalAnalysisConcurrency**
(from the analysis_concurrency specification in InterfIndControl)

- int evalServers
  (from the evaluation_servers specification in InterfIndControl)

- String evalScheduling
  evaluation_static_scheduling specifications in InterfIndControl)

- int analysisServers
  (from the analysis_servers specification in InterfIndControl)

- String analysisScheduling
  analysis_static_scheduling specifications in InterfIndControl)

- String failAction
  specification in InterfIndControl)

- int retryLimit
  retry specification in InterfIndControl)

- RealVector recoveryFnVals
  in InterfIndControl)

- bool activeSetVectorFlag
  InterfIndControl)

- bool evalCacheFlag
  specification in InterfIndControl)

- bool restartFileFlag
  deactivate restart_file specification in InterfIndControl)

Private Member Functions

- void assign (const DataInterface &data_interface)
  by copy constructor and assignment operator)

8.24.1 Detailed Description

Container class for interface specification data.

The DataInterface class is used to contain the data from an interface keyword specification. It is populated by ProblemDescDB::interface_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataInterface objects is maintained in ProblemDescDB::interfaceList, one for each interface specification.
in an input file. Default values are managed in the DataInterface constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::interfaceList is private (a similar model is used with SurrogateDataPoint objects contained in Dakota::Approximation).

The documentation for this class was generated from the following files:

- DataInterface.H
- DataInterface.C
8.25 DataMethod Class Reference

Container class for method specification data.

Public Member Functions

- DataMethod ()
  constructor
- DataMethod (const DataMethod &)
  copy constructor
- ~DataMethod ()
  destructor
- DataMethod & operator= (const DataMethod &)
  assignment operator
- bool operator== (const DataMethod &)
  equality operator
- void write (ostream &s) const
  write a DataMethod object to an 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

Public Attributes

- String idMethod
  the id_method specification in MethodIndControl
- String modelPointer
  (from the model_pointer specification in MethodIndControl)
- String methodOutput
  (default) (from the output specification in MethodIndControl)
8.25 DataMethod Class Reference

- int `maxIterations`
  \( \text{max\_iterations\ specification in MethodIndControl} \)
- int `maxFunctionEvaluations`
  \( \text{the max\_function\_evaluations\ specification in MethodIndControl} \)
- bool `speculativeFlag`
  \( \text{(from the speculative\ specification in MethodIndControl)} \)
- Real `convergenceTolerance`
  \( \text{convergence\_tolerance\ specification in MethodIndControl} \)
- Real `constraintTolerance`
  \( \text{constraint\_tolerance\ specification in MethodIndControl} \)
- bool `methodScaling`
  \( \text{MethodIndControl} \)
- RealVector `linearIneqConstraintCoeffs`
  \( \text{MethodIndControl).} \)
- RealVector `linearIneqLowerBnds`
  \( \text{linear\_inequality\_lower\_bounds\ specification in MethodIndControl} \)
- RealVector `linearIneqUpperBnds`
  \( \text{linear\_inequality\_upper\_bounds\ specification in MethodIndControl} \)
- StringArray `linearIneqScaleTypes`
  \( \text{linear\_inequality\_scale\_types\ specification in MethodIndControl} \)
- RealVector `linearIneqScales`
  \( \text{linear\_inequality\_scales\ specification in MethodIndControl} \)
- RealVector `linearEqConstraintCoeffs`
  \( \text{MethodIndControl).} \)
- RealVector `linearEqTargets`
  \( \text{linear\_equality\_targets\ specification in MethodIndControl} \)
- StringArray `linearEqScaleTypes`
  \( \text{linear\_equality\_scale\_types\ specification in MethodIndControl} \)
- RealVector `linearEqScales`
  \( \text{linear\_equality\_scales\ specification in MethodIndControl} \)
- String `methodName`
or parameter study methods

- **String minMaxType**
  
  *the optimization_type specification in MethodDOTDC*

- **String dlDetails**
  
  *string of options for a dynamically linked solver*

- **int verifyLevel**
  
  *the verify_level specification in MethodNPSOLDC*

- **Real functionPrecision**
  
  *the function_precision specification in MethodNPSOLDC*

- **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**
  
  *interior-point methods in MethodOPTPPDC*

- **Real gradientTolerance**
  
  *the gradient_tolerance specification in MethodOPTPPDC*
- Real `maxStep`  
  *the max_step specification in MethodOPTPPDC*

- String `meritFn`  
  *interior-point methods in MethodOPTPPDC*

- String `centralPath`  
  *methods in MethodOPTPPDC*

- Real `stepLenToBoundary`  
  *interior-point methods in MethodOPTPPDC*

- Real `centeringParam`  
  *interior-point methods in MethodOPTPPDC*

- int `searchSchemeSize`  
  *MethodOPTPPDC*

- String `evalSynchronization`  
  *methods in MethodCOLINYP and MethodCOLINYAPPS*

- Real `constraintPenalty`  
  *MethodCOLINYSW and MethodCOLINYEA*

- bool `constantPenalty`  
  *MethodCOLINYP and MethodCOLINYSW*

- Real `globalBalanceParam`  
  *MethodCOLINYDIR*

- Real `localBalanceParam`  
  *MethodCOLINYDIR*

- Real `maxBoxSize`  
  *the max_boxsize_limit for the DIRECT method in MethodCOLINYDIR*

- Real `minBoxSize`  
  *the min_boxsize_limit for the DIRECT method in MethodCOLINYDIR*

- Real `volBoxSize`  
  *the volume_boxsize_limit for the DIRECT method in MethodNCSUDC*

- String `boxDivision`  
  *the DIRECT method in MethodCOLINYDIR*
- `bool mutationAdaptive`
  \textit{MethodCOLINYEA}

- `bool showMiscOptions`
  \textit{the show\_misc\_options specification in MethodCOLINYDC}

- `StringArray miscOptions`
  \textit{the misc\_options specification in MethodCOLINYDC}

- `Real solnAccuracy`
  \textit{the solution\_accuracy specification in MethodCOLINYDC}

- `Real crossoverRate`
  \textit{the crossover\_rate specification for EA methods in MethodCOLINYEA}

- `Real mutationRate`
  \textit{the mutation\_rate specification for EA methods in MethodCOLINYEA}

- `Real mutationScale`
  \textit{the mutation\_scale specification for EA methods in MethodCOLINYEA}

- `Real mutationMinScale`
  \textit{MethodCOLINYEA}

- `Real initDelta`
  \textit{and MethodCOLINYSW}

- `Real threshDelta`
  \textit{and MethodCOLINYSW}

- `Real contractFactor`
  \textit{MethodCOLINYAPPS, MethodCOLINYP, and MethodCOLINYSW}

- `int newSolnsGenerated`
  \textit{in MethodCOLINYEA}

- `int numberRetained`
  \textit{MethodCOLINYEA.}

- `bool expansionFlag`
  \textit{MethodCOLINYAPPS, MethodCOLINYP, and MethodCOLINYSW}

- `int expandAfterSuccess`
  \textit{MethodCOLINYP and MethodCOLINYSW}

- `int contractAfterFail`
MethodCOLINYSW

- int mutationRange

MethodCOLINYEA

- int totalPatternSize

MethodCOLINYPS.

- bool randomizeOrderFlag

MethodCOLINYPS.

- String selectionPressure

  *the fitness_type specification for EA methods in MethodCOLINYEA*

MethodCOLINYEA

- String replacementType

  *MethodCOLINYEA.*

- String crossoverType

  *the crossover_type specification for EA methods in MethodCOLINYEA*

- String mutationType

  *the mutation_type specification for EA methods in MethodCOLINYEA*

- String exploratoryMoves

  *MethodCOLINYPS.*

- String patternBasis

  *MethodCOLINYAPPS and MethodCOLINYPS.*

- 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

  *for a fitness tracker converger.*
- size_t numGenerations
  tracker converger should track.

- Real fitnessLimit
  below_limit selector).

- Real shrinkagePercent
  must take place on each call to the selector (0, 1).

- String nichingType
  The niching type.

- RealVector nicheVector
  The discretization percentage along each objective.

- 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
  MethodCOLINYEA.

- bool printPopFlag
  at each generation

- String daceMethod
  dace specification in MethodDDACE)

- int numSymbols
  the symbols specification for DACE methods

- bool mainEffectsFlag
  in MethodDDACE)
8.25 DataMethod Class Reference

- bool latinizeFlag
  MethodFSUDACE

- bool volQualityFlag
  and CVT methods in MethodFSUDACE

- bool varBasedDecompFlag
  and CVT methods in MethodFSUDACE

- IntVector sequenceStart
  the sequenceStart specification in MethodFSUDACE

- IntVector sequenceLeap
  the sequenceLeap specification in MethodFSUDACE

- IntVector primeBase
  the primeBase specification in MethodFSUDACE

- int numTrials
  the numTrials specification in MethodFSUDACE

- String trialType
  the trial_type specification in MethodFSUDACE

- int randomSeed
  the seed specification for COLINY, NonD, & DACE methods

- int numSamples
  the samples specification for NonD & DACE methods

- bool fixedSeedFlag
  stencil/pattern throughout a strategy with repeated sampling.

- bool fixedSequenceFlag
  stencil/pattern throughout a strategy with repeated sampling.

- int previousSamples
  the number of previous samples when augmenting a LHS sample

- int expansionTerms
  the expansion_terms specification in MethodNonDPCE

- short expansionOrder
  the expansion_order specification in MethodNonDPCE

- int expansionSamples
the expansion_samples specification in MethodNonDPCE

- **ShortArray quadratureOrder**
  the quadrature_order specification in MethodNonDPCE

- **short cubatureLevel**
  the cubature_level specification in MethodNonDPCE

- **int collocationPoints**
  the collocation_points specification in MethodNonDPCE

- **String expansionImportFile**
  the expansion_import_file specification in MethodNonDPCE

- **String sampleType**
  MethodNonDPCE.

- **String reliabilitySearchType**
  MethodNonDGlobalRel(x_gaussian_process or u_gaussian_process)

- **String reliabilitySearchAlgorithm**
  by sqp or nip in MethodNonDLocalRel

- **String reliabilityIntegration**
  MethodNonDLocalRel

- **String reliabilityIntegrationRefine**
  integration refinement selection in MethodNonDLocalRel

- **String distributionType**
  and MethodNonDGlobalRel

- **String responseLevelMappingType**
  MethodNonDLocalRel, and MethodNonDGlobalRel

- **RealVectorArray responseLevels**
  MethodNonDPCE, MethodNonDLocalRel, and MethodNonDGlobalRel

- **RealVectorArray probabilityLevels**
  MethodNonDPCE, MethodNonDLocalRel, and MethodNonDGlobalRel

- **RealVectorArray reliabilityLevels**
  MethodNonDPCE, and MethodNonDLocalRel

- **RealVectorArray genReliabilityLevels**
  MethodNonDPCE, MethodNonDLocalRel, and MethodNonDGlobalRel
8.25 DataMethod Class Reference

- `bool allVarsFlag`
  
  *the all_variables specification in MethodNonDMC*

- `short paramStudyType`
  
  *centered(4), or multidim(5)*

- `RealVector finalPoint`
  
  *the final_point specification in MethodPSVPS*

- `RealVector stepVector`
  
  *the step_vector specification in MethodPSVPS*

- `Real stepLength`
  
  *the step_length specification in MethodPSVPS*

- `int numSteps`
  
  *the num_steps specification in MethodPSVPS*

- `RealVector listOfPoints`
  
  *the list_of_points specification in MethodPSLPS*

- `Real percentDelta`
  
  *the percent_delta specification in MethodPSCPS*

- `int deltasPerVariable`
  
  *the deltas_per_variable specification in MethodPSCPS*

- `IntArray varPartitions`
  
  *MethodPSMPS.*

Private Member Functions

- `void assign (const DataMethod &data_method)`
  
  *by copy constructor and assignment operator*

8.25.1 Detailed Description

Container class for method specification data.

The DataMethod class is used to contain the data from a method keyword specification. It is populated by ProblemDescDB::method_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataMethod objects is maintained in ProblemDescDB::methodList, one for each method specification in an input file. Default values are managed in the DataMethod constructor. Data is public to avoid maintaining
set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::methodList is private (a similar model is used with SurrogateDataPoint objects contained in Dakota::Approximation).

The documentation for this class was generated from the following files:

- DataMethod.H
- DataMethod.C
8.26 DataModel Class Reference

Container class for model specification data.

Public Member Functions

- **DataModel ()**
  constructor

- **DataModel (const DataModel &)**
  copy constructor

- **~DataModel ()**
  destructor

- **DataModel & operator= (const DataModel &)**
  assignment operator

- **bool operator== (const DataModel &)**
  equality operator

- **void write (ostream &s) const**
  write a DataModel object to an 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

Public Attributes

- **String idModel**
  the id_model specification in ModelIndControl

- **String modelType**
  specification in ModelIndControl

- **String variablesPointer**
  (from the variables_pointer specification in ModelIndControl)
- **String interfacePointer**
  *the optional_interface_pointer specification in ModelNested*

- **String responsesPointer**
  *(from the responses_pointer specification in ModelIndControl)*

- **IntSet surrogateFnIndices**
  *array specifying the response function set that is approximated*

- **String surrogateType**
  *polynomial, kriging, or hierarchical*

- **String actualModelPtr**
  *and ModelSurrMP*

- **String lowFidelityModelPtr**
  *specification in ModelSurrH*

- **String highFidelityModelPtr**
  *specification in ModelSurrH*

- **String approxDaceMethodPtr**
  *specification in ModelSurrG*

- **String approxSampleReuse**
  *ModelSurrG*

- **String approxSampleReuseFile**
  *specification in ModelSurrG*

- **String approxCorrectionType**
  *in ModelSurrG and ModelSurrH*

- **short approxCorrectionOrder**
  *and ModelSurrH*

- **bool approxGradUsageFlag**
  *(from the use_gradients specification in ModelSurrG)*

- **RealVector krigingCorrelations**
  *(from the correlations specification in ModelSurrG)*

- **short polynomialOrder**
  *in ModelSurrG*

- **short trendOrder**
gaussian_process specification in ModelSurrG)

- **bool** pointSelection
  flag indicating the use of point selection in the Gaussian process

- **String** optionalInterfRespPointer
  optional_interface_responses_pointer specification in ModelNested)

- **String** subMethodPointer
  the sub_method_pointer specification in ModelNested)

- **StringArray** primaryVarMaps
  ModelNested)

- **StringArray** secondaryVarMaps
  secondary_variable_mapping specification in ModelNested)

- **RealVector** primaryRespCoeffs
  specification in ModelNested)

- **RealVector** secondaryRespCoeffs
  specification in ModelNested)

**Private Member Functions**
- **void** assign (const DataModel &data_model)
  by copy constructor and assignment operator)

### 8.26.1 Detailed Description

Container class for model specification data.

The **DataModel** class is used to contain the data from a model keyword specification. It is populated by ProblemDescDB::model_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of **DataModel** objects is maintained in ProblemDescDB::modelList, one for each model specification in an input file. Default values are managed in the **DataModel** constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::modelList is private (a similar model is used with SurrogateDataPoint objects contained in Dakota::Approximation).

The documentation for this class was generated from the following files:

- DataModel.H
- DataModel.C
8.27 DataResponses Class Reference

Container class for responses specification data.

Public Member Functions

- **DataResponses ()**
  
  *constructor*

- **DataResponses (const DataResponses &)**

  *copy constructor*

- **~DataResponses ()**

  *destructor*

- **DataResponses & operator= (const DataResponses &)**

  *assignment operator*

- **bool operator==(const DataResponses &)**

  *equality operator*

- **void write (ostream &s) const**

  *write a DataResponses object to an 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*

Public Attributes

- **size_t numObjectiveFunctions**

  *num_objective_functions specification in RespFnOpt*

- **size_t numNonlinearIneqConstraints**

  *num_nonlinear_inequality_constraints specification in RespFnOpt*

- **size_t numNonlinearEqConstraints**

  *num_nonlinear_equality_constraints specification in RespFnOpt*
- `size_t numLeastSqTerms
  num_least_squares_terms specification in RespFnLS

- `size_t numResponseFunctions
  num_response_functions specification in RespFnGen

- `StringArray objectiveFunctionScaleTypes
  objective_function_scale_types specification in RespFnOpt

- `RealVector objectiveFunctionScales
  objective_function_scales specification in RespFnOpt

- `RealVector multiObjectiveWeights
  multi_objective_weights specification in RespFnOpt

- `String leastSqDataFile
  RespFnOpt

- `StringArray leastSqTermScaleTypes
  least_squares_term_scale_types specification in RespFnOpt

- `RealVector leastSqTermScales
  least_squares_term_scales specification in RespFnOpt

- `RealVector nonlinearIneqLowerBnds
  nonlinear_inequality_lower_bounds specification in RespFnOpt

- `RealVector nonlinearIneqUpperBnds
  nonlinear_inequality_upper_bounds specification in RespFnOpt

- `StringArray nonlinearIneqScaleTypes
  nonlinear_inequality_scale_types specification in RespFnOpt

- `RealVector nonlinearIneqScales
  nonlinear_inequality_scales specification in RespFnOpt

- `RealVector nonlinearEqTargets
  nonlinear_equality_targets specification in RespFnOpt

- `StringArray nonlinearEqScaleTypes
  nonlinear_equality_scale_types specification in RespFnOpt

- `RealVector nonlinearEqScales
  nonlinear_equality_scales specification in RespFnOpt

- `String gradientType
mixed_gradients specifications in RespGrad)

- **String hessianType**
  
  RespHess)

- **String quasiHessianType**
  
  and sr1 specifications in RespHess)

- **String methodSource**
  
  method_source specification in RespGradNum and RespGradMixed)

- **String intervalType**
  
  interval_type specification in RespGradNum and RespGradMixed)

- **RealVector fdGradStepSize**
  
  specification in RespGradNum and RespGradMixed)

- **RealVector fdHessStepSize**
  
  RespHessMixed)

- **IntList idNumericalGrads**
  
  specification in RespGradMixed)

- **IntList idAnalyticGrads**
  
  specification in RespGradMixed)

- **IntList idNumericalHessians**
  
  specification in RespHessMixed)

- **IntList idQuasiHessians**
  
  specification in RespHessMixed)

- **IntList idAnalyticHessians**
  
  specification in RespHessMixed)

- **String idResponses**
  
  (from the id_responses specification in RespSetId)

- **StringArray responseLabels**
  
  specification in RespLabels)

### Private Member Functions

- **void assign (const DataResponses &data_responses)**
  
  by copy constructor and assignment operator)
8.27.1 Detailed Description

Container class for responses specification data.

The DataResponses class is used to contain the data from a responses keyword specification. It is populated by
ProblemDescDB::responses_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A
list of DataResponses objects is maintained in ProblemDescDB::responsesList, one for each responses specifi-
cation in an input file. Default values are managed in the DataResponses constructor. Data is public to avoid
maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::responses-
List is private (a similar model is used with SurrogateDataPoint objects contained in Dakota::Approximation).

The documentation for this class was generated from the following files:

- DataResponses.H
- DataResponses.C
8.28 DataStrategy Class Reference

Container 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 (ostream &s) const**
  write a DataStrategy object to an 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

Public Attributes

- **String strategyType**
  branch_and_bound, multi_start, pareto_set, or single_method

- **bool graphicsFlag**
  specification in StratIndControl)

- **bool tabularDataFlag**
  the tabular_graphics_data specification in StratIndControl)

- **String tabularDataFile**
  the tabular_graphics_file specification in StratIndControl)
- int iteratorServers
  the iterator_servers specification in StratIndControl

- String iteratorScheduling
  iterator_static_scheduling specifications in StratIndControl

- String methodPointer
  StratMultiStart).

- StringArray multilevelMethodList
  strategy (from the method_list specification in StratML)

- String multilevelType
  adaptive_hybrid, and coupled specifications in StratML

- Real multilevelProgThresh
  (from the progress_threshold specification in StratML)

- String multilevelGlobalMethodPointer
  (from the global_method_pointer specification in StratML)

- String multilevelLocalMethodPointer
  (from the local_method_pointer specification in StratML)

- Real multilevelLSProb
  (from the local_search_probability specification in StratML)

- size_t numSolutionsTransferred
  to a subsequent method the multilevel strategy

- int surrBasedOptMaxIterations
  in StratSBO

- Real surrBasedOptConvTol
  (from the convergence_tolerance specification in StratSBO)

- int surrBasedOptSoftConvLimit
  specification in StratSBO)

- bool surrBasedOptLayerBypass
  in evaluating truth response values in SBO.

- Real surrBasedOptTRInitSize
  distance (upper bound - lower bound) for each variable

- Real surrBasedOptTRMinSize
regions)

- Real `surrBasedOptTRContractTrigger`
  
  ("eta_L" in the Conn-Gould-Toint trust region book)

- Real `surrBasedOptTRExpandTrigger`
  

- Real `surrBasedOptTRContract`
  
  specification in StratSBO)

- Real `surrBasedOptTRExpand`
  
  in StratSBO)

- short `surrBasedOptSubProbObj`
  
  LAGRANGIAN_OBJECTIVE, or AUGMENTED_LAGRANGIAN_OBJECTIVE.

- short `surrBasedOptSubProbCon`
  
  LINEARIZED_CONSTRAINTS, or ORIGINAL_CONSTRAINTS.

- short `surrBasedOptMeritFn`
  
  BASIC_LAGRANGIAN, or AUGMENTED_LAGRANGIAN.

- short `surrBasedOptAcceptLogic`
  
  SBO iterate acceptance logic: TR_RATIO or FILTER.

- short `surrBasedOptConstrRelax`
  
  SBO constraint relaxation method: NO_RELAX or HOMOTOPY.

- int `concurrentRandomJobs`
  
  in StratMultiStart and StratParetoSet)

- int `concurrentSeed`
  
  and StratParetoSet)

- RealVector `concurrentParameterSets`
  
  StratMultiStart and StratParetoSet).

**Private Member Functions**

- void `assign` (const DataStrategy &data_strategy)
  
  by copy constructor and assignment operator)
8.28 DataStrategy Class Reference

8.28.1 Detailed Description

Container class for strategy specification data.

The DataStrategy class is used to contain the data from a strategy keyword specification. It is populated by
ProblemDescDB::strategy_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. De-
fault values are managed in the DataStrategy constructor. Data is public to avoid maintaining set/get functions,
but is still encapsulated within ProblemDescDB since ProblemDescDB::strategySpec is private (a similar model
is used with SurrogateDataPoint objects contained in Dakota::Approximation).

The documentation for this class was generated from the following files:

- DataStrategy.H
- DataStrategy.C
8.29 DataVariables Class Reference

Container 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 (ostream &s) const**
  *write a DataVariables object to an 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 uncertain ()**
  *return total number of uncertain variables*

- **size_t state ()**
  *return total number of state variables*

- **size_t num_continuous_variables ()**
  *return total number of continuous variables*
8.29 DataVariables Class Reference

- **size_t num_discrete_variables ()**
  
  return total number of discrete variables

- **size_t num_variables ()**
  
  return total number of variables

### Public Attributes

- **String idVariables**
  
  (from the id_variables specification in VarSetId)

- **size_t numContinuousDesVars**
  
  specification in VarDV)

- **size_t numDiscreteDesVars**
  
  specification in VarDV)

- **size_t numNormalUncVars**
  
  specification in VarUV)

- **size_t numLognormalUncVars**
  
  specification in VarUV)

- **size_t numUniformUncVars**
  
  specification in VarUV)

- **size_t numLoguniformUncVars**
  
  loguniform_uncertain specification in VarUV)

- **size_t numTriangularUncVars**
  
  triangular_uncertain specification in VarUV)

- **size_t numExponentialUncVars**
  
  exponential_uncertain specification in VarUV)

- **size_t numBetaUncVars**
  
  specification in VarUV)

- **size_t numGammaUncVars**
  
  specification in VarUV)

- **size_t numGumbelUncVars**
  
  specification in VarUV)

- **size_t numFrechetUncVars**
• size_t numWeibullUncVars
  specification in VarUV
• size_t numHistogramUncVars
  specification in VarUV
• size_t numIntervalUncVars
  specification in VarUV
• size_t numContinuousStateVars
  specification in VarSV
• size_t numDiscreteStateVars
  specification in VarSV
• RealVector continuousDesignVars
  the cdv_initial_point specification in VarDV
• RealVector continuousDesignLowerBnds
  cdv_lower_bounds specification in VarDV
• RealVector continuousDesignUpperBnds
  cdv_upper_bounds specification in VarDV
• StringArray continuousDesignScaleTypes
  cdv_scale_types specification in VarDV
• RealVector continuousDesignScales
  cdv_scales specification in VarDV
• IntVector discreteDesignVars
  the ddv_initial_point specification in VarDV
• IntVector discreteDesignLowerBnds
  ddv_lower_bounds specification in VarDV
• IntVector discreteDesignUpperBnds
  ddv_upper_bounds specification in VarDV
• StringArray continuousDesignLabels
  specification in VarDV
• StringArray discreteDesignLabels
  specification in VarDV
- **RealVector normalUncMeans**
  *specification in VarUV*

- **RealVector normalUncStdDevs**
  *the nuv_std_deviations specification in VarUV*

- **RealVector normalUncLowerBnds**
  *(from the nuv_lower_bounds specification in VarUV)*

- **RealVector normalUncUpperBnds**
  *(from the nuv_upper_bounds specification in VarUV)*

- **RealVector lognormalUncMeans**
  *lnuv_means specification in VarUV*

- **RealVector lognormalUncStdDevs**
  *the lnuv_std_deviations specification in VarUV*

- **RealVector lognormalUncErrFacts**
  *the lnuv_error_factors specification in VarUV*

- **RealVector lognormalUncLowerBnds**
  *(from the lnuv_lower_bounds specification in VarUV)*

- **RealVector lognormalUncUpperBnds**
  *(from the lnuv_upper_bounds specification in VarUV)*

- **RealVector uniformUncLowerBnds**
  *(from the uuv_lower_bounds specification in VarUV)*

- **RealVector uniformUncUpperBnds**
  *(from the uuv_upper_bounds specification in VarUV)*

- **RealVector loguniformUncLowerBnds**
  *(from the luuv_lower_bounds specification in VarUV)*

- **RealVector loguniformUncUpperBnds**
  *(from the luuv_upper_bounds specification in VarUV)*

- **RealVector triangularUncModes**
  *specification in VarUV*

- **RealVector triangularUncLowerBnds**
  *(from the tuv_lower_bounds specification in VarUV)*
- `RealVector triangularUncUpperBnds`  
  \((from \ the \ tuv\_upper\_bounds \ specification \ in \ VarUV)\)

- `RealVector exponentialUncBetas`  
  \((the \ euv\_betas \ specification \ in \ VarUV)\)

- `RealVector betaUncAlphas`  
  \((the \ buv\_means \ specification \ in \ VarUV)\)

- `RealVector betaUncBetas`  
  \((the \ buv\_std\_deviations \ specification \ in \ VarUV)\)

- `RealVector betaUncLowerBnds`  
  \((from \ the \ buv\_lower\_bounds \ specification \ in \ VarUV)\)

- `RealVector betaUncUpperBnds`  
  \((from \ the \ buv\_upper\_bounds \ specification \ in \ VarUV)\)

- `RealVector gammaUncAlphas`  
  \((the \ gauv\_alphas \ specification \ in \ VarUV)\)

- `RealVector gammaUncBetas`  
  \((the \ gauv\_betas \ specification \ in \ VarUV)\)

- `RealVector gumbelUncAlphas`  
  \((guuv\_alphas \ specification \ in \ VarUV)\)

- `RealVector gumbelUncBetas`  
  \((the \ guuv\_betas \ specification \ in \ VarUV)\)

- `RealVector frechetUncAlphas`  
  \((the \ fuv\_alphas \ specification \ in \ VarUV)\)

- `RealVector frechetUncBetas`  
  \((the \ fuv\_betas \ specification \ in \ VarUV)\)

- `RealVector weibullUncAlphas`  
  \((the \ wuv\_alphas \ specification \ in \ VarUV)\)

- `RealVector weibullUncBetas`  
  \((the \ wuv\_betas \ specification \ in \ VarUV)\)

- `RealVectorArray histogramUncBinPairs`  
  \((specifications \ in \ VarUV)\)

- `RealVectorArray histogramUncPointPairs`
- RealVectorArray intervalUncBasicProbs
  \textit{intervalUncBasicProbs} specification in \texttt{VarUV})

- RealVectorArray intervalUncBounds
  \textit{intervalUncBounds} specification in \texttt{VarUV})

- RealMatrix uncertainCorrelations
  \textit{uncertainCorrelations} matrix for analytic reliability methods.

- RealVector uncertainVars
  \textit{uncertainVars} initialized in \texttt{IDRProblemDescDB::variables_kwhandler()}

- RealVector uncertainLowerBnds
  \textit{uncertainLowerBnds} for gamma, gumbel, frechet, weibull and histogram specifications)

- RealVector uncertainUpperBnds
  \textit{uncertainUpperBnds} for gamma, gumbel, frechet, weibull and histogram specifications)

- StringArray uncertainLabels
  \textit{uncertainLabels} \textit{huv_descriptors} specifications in \texttt{VarUV})

- RealVector continuousStateVars
  \textit{continuousStateVars} \textit{csv_initial_state} specification in \texttt{VarSV})

- RealVector continuousStateLowerBnds
  \textit{continuousStateLowerBnds} \textit{csv_lower_bounds} specification in \texttt{VarSV})

- RealVector continuousStateUpperBnds
  \textit{continuousStateUpperBnds} \textit{csv_upper_bounds} specification in \texttt{VarSV})

- IntVector discreteStateVars
  \textit{discreteStateVars} \textit{dsv_initial_state} specification in \texttt{VarSV})

- IntVector discreteStateLowerBnds
  \textit{discreteStateLowerBnds} \textit{dsv_lower_bounds} specification in \texttt{VarSV})

- IntVector discreteStateUpperBnds
  \textit{discreteStateUpperBnds} \textit{dsv_upper_bounds} specification in \texttt{VarSV})

- StringArray continuousStateLabels
  \textit{continuousStateLabels} \textit{specification in VarSV})

- StringArray discreteStateLabels
  \textit{discreteStateLabels} \textit{specification in VarSV})
Private Member Functions

- void assign (const DataVariables &data_variables)
  
  \textit{by copy constructor and assignment operator}

8.29.1 Detailed Description

Container class for variables specification data.

The \texttt{DataVariables} class is used to contain the data from a variables keyword specification. It is populated by \texttt{IDRProblemDescDB::variables_kwhandler()} and is queried by the \texttt{ProblemDescDB::get_<datatype>()} functions. A list of \texttt{DataVariables} objects is maintained in \texttt{ProblemDescDB::variablesList}, one for each variables specification in an input file. Default values are managed in the \texttt{DataVariables} constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within \texttt{ProblemDescDB} since \texttt{ProblemDescDB::variablesList} is private (a similar model is used with \texttt{SurrogateDataPoint} objects contained in \texttt{Dakota::Approximation}).

The documentation for this class was generated from the following files:

- \texttt{DataVariables.H}
- \texttt{DataVariables.C}
8.30 DDACEDesignCompExp Class Reference

Wrapper class for the DDACE design of experiments library.

Inheritance diagram for DDACEDesignCompExp:

```
  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 extract_trends ()**  
  Redefines the run_iterator virtual function for the PStudy/DACE branch.

- **void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)**  
  reset sampling iterator

- **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 (const 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()`
  number of symbols from input.

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`
  (inversely related to number of replications)

- const int `originalSeed`
  (allows repeatable results)

- int `randomSeed`
  current seed for the random number generator

- bool `allDataFlag`
  `Iterator::all_variables()` and `Iterator::all_responses()`.

- size_t `numDACERuns`
  counter for number of `run()` executions for this object

- bool `varyPattern`
  multiple executions are repeatable but not correlated.

- bool `volQualityFlag`
  flag which specifies evaluating the volumetric quality measures

- bool `varBasedDecompFlag`
  flag which specifies variance based decomposition
8.30 DDACEDesignCompExp Class Reference

- bool mainEffectsFlag
  flag which specifies main effects

- std::vector< std::vector< int > > symbolMapping
  mapping of symbols for main effects calculations

8.30.1 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.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 DDACEDesignCompExp (Model & model)

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

8.30.2.2 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.

8.30.3 Member Function Documentation

8.30.3.1 void resolve_samples_symbols () [private]

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.
The documentation for this class was generated from the following files:

- DDACEDesignCompExp.H
- DDACEDesignCompExp.C
DirectFnApplicInterface Class Reference

Sample derived interface class for testing plug-ins using `assign_rep()`.

Inheritance diagram for DirectFnApplicInterface:

```
  Interface
   
  ApplicationInterface
   
  DirectFnApplicInterface
   
  DirectFnApplicInterface
```

Public Member Functions

- **DirectFnApplicInterface** (const Dakota::ProblemDescDB &problem_db)
  
  *constructor*

- **~DirectFnApplicInterface** ()

  *destructor*

Protected Member Functions

- int **derived_map_ac** (const Dakota::String &ac_name)

  *execute an analysis code portion of a direct evaluation invocation*

8.31.1 Detailed Description

Sample derived interface class for testing plug-ins using `assign_rep()`.

The plug-in `DirectFnApplicInterface` resides in namespace `SIM` and uses a copy of `rosenbrock()` to perform 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.C` (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".

The documentation for this class was generated from the following files:
- PluginDirectFnApplicInterface.H
- PluginDirectFnApplicInterface.C
8.32 DirectFnApplicInterface Class Reference

and testers using direct procedure calls.

Inheritance diagram for DirectFnApplicInterface:

```
<table>
<thead>
<tr>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationInterface</td>
</tr>
<tr>
<td>DirectFnApplicInterface</td>
</tr>
<tr>
<td>DirectFnApplicInterface</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **DirectFnApplicInterface** (const ProblemDescDB &problem_db)  
  *constructor*

- **~DirectFnApplicInterface** ()  
  *destructor*

- void **derived_map** (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)  
  *that is specific to a derived class.*

- void **derived_map_asynch** (const ParamResponsePair &pair)  
  *asynchronous evaluation that is specific to a derived class.*

- void **derived_synch** (PRPList &prp_list)  
  *classes. This version waits for at least one completion.*

- void **derived_synch_nowait** (PRPList &prp_list)  
  *any completions if none are immediately available.*

- int **derived_synchronous_local_analysis** (const int &analysis_id)

- const **StringArray & analysis_drivers** () const  
  *retrieve the analysis drivers specification for application interfaces*
Protected Member Functions

- virtual int derived_map_if (const String &if_name)
  execute the input filter portion of a direct evaluation invocation

- virtual int derived_map_ac (const String &ac_name)
  execute an analysis code portion of a direct evaluation invocation

- virtual int derived_map_of (const String &of_name)
  execute the output filter portion of a direct evaluation invocation

- void set_local_data (const Variables &vars, const ActiveSet &set, const Response &response)
  variable attributes and zeros response data

- void overlay_response (Response &response)
  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

- 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 numADV
  total number of discrete variables

- size_t numDerivVars
  number of active derivative variables
- `RealVector xC
  continuous variables used within direct simulator fns`

- `IntVector xD
  discrete variables used within direct simulator fns`

- `StringArray xCLabels
  continuous variable labels`

- `StringArray xDLabels
  discrete variable labels`

- `ShortArray directFnASV
  class scope active set vector`

- `IntArray directFnDVV
  class scope derivative variables vector`

- `RealVector fnVals
  response fn values within direct simulator fns`

- `RealMatrix fnGrads
  response fn gradients w/i direct simulator fns`

- `RealMatrixArray fnHessians
  response fn Hessians w/i direct simulator fns`

- `StringArray fnLabels
  response function labels`

- `StringArray analysisDrivers
  (from the analysis_components interface specification)`

- `size_t analysisDriverIndex
  the index of the active analysis driver within analysisDrivers`

- `String2DArray analysisComponents
  (from the analysis_components interface specification)`

- `engine * matlabEngine
  pointer to the MATLAB engine used for direct evaluations`
Private Member Functions

- int cantilever ()
  the cantilever UQ/OUU test function

- int cyl_head ()
  the cylinder head constrained optimization test fn

- 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 log_ratio ()
  the log_ratio UQ test function

- int short_column ()
  the short_column UQ/OUU test function

- int steel_column_cost ()
  the steel_column_cost UQ/OUU test function

- int steel_column_perf ()
  the short_column_perf UQ/OUU 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 multimodal ()
  multimodal UQ test function
### 8.32 DirectFnApplicInterface Class Reference

- **int salinas ()**
  
  *direct interface to the SALINAS structural dynamics code*

- **int mc_api_run ()**
  
  *direct interface to ModelCenter via API, HKIM 4/3/03*

- **int matlab_engine_run ()**
  
  *direct interface to Matlab via API, BMA 11/28/05*

- **int python_run ()**
  
  *direct interface to Python via API, BMA 07/02/07*

- **template<class ArrayT> bool python_convert_int (const ArrayT &src, PyObject **dst)**
  
  *convert arrays of integer types to Python*

- **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 &d_src, PyObject **dst)**
  
  *convert RealVector + IntVector to Python mixed list or numpy double array*

- **bool python_convert (const StringArray &src, PyObject **dst)**
  
  *convert labels*

- **bool python_convert (const StringArray &c_src, const StringArray &d_src, PyObject **dst)**
  
  *convert all labels to single list*

- **bool python_convert (PyObject *pyv, RealBaseVector &rv, const int &dim)**
  
  *convert python list of int or float to RealVector*

- **bool python_convert (PyObject *pym, RealMatrix &rm)**
  
  *convert python list of lists of int or float to RealMatrix*

- **bool python_convert (PyObject *pyma, RealMatrixArray &rma)**
  
  *convert python list of lists of lists of int or float to RealMatrixArray*

### Private Attributes

- **bool userNumpyFlag**
  
  *whether the user requested numpy data structures*
8.32.1 Detailed Description

and testers using direct procedure calls.  
DirectFnApplicInterface uses a few linkable simulation codes and several internal member functions to perform parameter to response mappings.

8.32.2 Member Function Documentation

8.32.2.1 int derived_synchronous_local_analysis (const int & analysis_id) [inline, virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().  Reimplemented from ApplicationInterface.

8.32.2.2 int derived_map_ac (const 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 DirectFnApplicInterface.

The documentation for this class was generated from the following files:

- DirectFnApplicInterface.H
- DirectFnApplicInterface.C
8.33 DistinctConstraints Class Reference

the default data view (no variable or domain type array merging).

Inheritance diagram for DistinctConstraints::

```
Constraints  VariablesUtil
          |                   |
          |                   |
          |                   |
DistinctConstraints
```

Public Member Functions

- **DistinctConstraints ()**
  *default constructor*

- **DistinctConstraints (const ProblemDescDB &problem_db, const pair< short, short > &view)**
  *standard constructor*

- **~DistinctConstraints ()**
  *destructor*

- **const RealVector & continuous_lower_bounds () const**
  *return the active continuous variable lower bounds*

- **void continuous_lower_bounds (const RealVector &c_l_bnds)**
  *set the active continuous variable lower bounds*

- **const RealVector & continuous_upper_bounds () const**
  *return the active continuous variable upper bounds*

- **void continuous_upper_bounds (const RealVector &c_u_bnds)**
  *set the active continuous variable upper bounds*

- **const IntVector & discrete_lower_bounds () const**
  *return the active discrete variable lower bounds*

- **void discrete_lower_bounds (const IntVector &d_l_bnds)**
  *set the active discrete variable lower bounds*

- **const IntVector & discrete_upper_bounds () const**
  *return the active discrete variable upper bounds*
- void **discrete_upper_bounds** (const **IntVector** \&d_u_bnds)
  
  *set the active discrete variable upper bounds*

- const **RealVector** & **inactive_continuous_lower_bounds** () const
  
  *return the inactive continuous lower bounds*

- void **inactive_continuous_lower_bounds** (const **RealVector** \&i_c_l_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** \&i_c_u_bnds)
  
  *set the inactive continuous upper bounds*

- const **IntVector** & **inactive_discrete_lower_bounds** () const
  
  *return the inactive discrete lower bounds*

- void **inactive_discrete_lower_bounds** (const **IntVector** \&i_d_l_bnds)
  
  *set the inactive discrete lower bounds*

- const **IntVector** & **inactive_discrete_upper_bounds** () const
  
  *return the inactive discrete upper bounds*

- void **inactive_discrete_upper_bounds** (const **IntVector** \&i_d_u_bnds)
  
  *set the inactive discrete upper bounds*

- **RealVector** **all_continuous_lower_bounds** () const
  
  *returns a single array with all continuous lower bounds*

- void **all_continuous_lower_bounds** (const **RealVector** \&a_c_l_bnds)
  
  *sets all continuous lower bounds using a single array*

- **RealVector** **all_continuous_upper_bounds** () const
  
  *returns a single array with all continuous upper bounds*

- void **all_continuous_upper_bounds** (const **RealVector** \&a_c_u_bnds)
  
  *sets all continuous upper bounds using a single array*

- **IntVector** **all_discrete_lower_bounds** () const
  
  *returns a single array with all discrete lower bounds*

- void **all_discrete_lower_bounds** (const **IntVector** \&a_d_l_bnds)
  
  *sets all discrete lower bounds using a single array*
DistinctConstraints Class Reference

- **IntVector all_discrete_upper_bounds() const**
  returns a single array with all discrete upper bounds

- **void all_discrete_upper_bounds(const IntVector &a_d_u_bnds)**
  sets all discrete upper bounds using a single array

- **void write(ostream &s) const**
  write a variable constraints object to an ostream

- **void read(istream &s)**
  read a variable constraints object from an istream

Protected Member Functions

- **void copy_rep(const Constraints *con_rep)**
  Used by copy() to copy the contents of a letter class.

- **void reshape_rep(const Sizet2DArray &vars_comps)**
  Used by reshape(Sizet2DArray&) to rehape the contents of a letter class.

Private Attributes

- **RealVector continuousDesignLowerBnds**
  the continuous design lower bounds array

- **RealVector continuousDesignUpperBnds**
  the continuous design upper bounds array

- **IntVector discreteDesignLowerBnds**
  the discrete design lower bounds array

- **IntVector discreteDesignUpperBnds**
  the discrete design upper bounds array

- **RealVector uncertainLowerBnds**
  the uncertain distribution lower bounds array

- **RealVector uncertainUpperBnds**
  the uncertain distribution upper bounds array

- **RealVector continuousStateLowerBnds**
  the continuous state lower bounds array
8.33.1 Detailed Description

the default data view (no variable or 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 DistinctConstraints derived class separates the design, uncertain, and state variable types as well as the continuous and discrete domain types. The result is separate lower and upper bounds arrays for continuous design, discrete design, uncertain, continuous state, and discrete state variables. This is the default approach, so all iterators and strategies not specifically utilizing the All or Merged views use this approach (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to the Constraints constructor in Model).

8.33.2 Constructor & Destructor Documentation

8.33.2.1 DistinctConstraints (const ProblemDescDB & problem_db, const pair< short, short > & view)

standard constructor

In this class, the distinct approach (design, uncertain, and state types are distinct) is used. Most iterators/strategies use this approach, which is the default in Constraints::get_constraints(). Extract distinct lower and upper bounds (VariablesUtil is not used).

The documentation for this class was generated from the following files:

- DistinctConstraints.H
- DistinctConstraints.C
8.34 DistinctVariables Class Reference

the default data view (no variable or domain type array merging).

Inheritance diagram for DistinctVariables::

```
DistinctVariables
  |________________________|
  | VariablesUtil          |
  |________________________|
  | Variables               |
```

Public Member Functions

- `DistinctVariables ()`  
  *default constructor*

- `DistinctVariables (const ProblemDescDB &problem_db, const pair< short, short > &view)`  
  *standard constructor*

- `~DistinctVariables ()`  
  *destructor*

- `size_t tv () const`  
  *Returns total number of vars.*

- `const RealVector & continuous_variables () const`  
  *return the active continuous variables*

- `void continuous_variables (const RealVector &c_vars)`  
  *set the active continuous variables*

- `const IntVector & discrete_variables () const`  
  *return the active discrete variables*

- `void discrete_variables (const IntVector &d_vars)`  
  *set the active discrete variables*

- `const StringArray & continuous_variable_labels () const`  
  *return the active continuous variable labels*

- `void continuous_variable_labels (const StringArray &c_v_labels)`  
  *set the active continuous variable labels*
• const StringArray & discrete_variable_labels () const
  return the active discrete variable labels

• void discrete_variable_labels (const StringArray &d_v_labels)
  set the active discrete variable labels

• const RealVector & inactive_continuous_variables () const
  return the inactive continuous variables

• void inactive_continuous_variables (const RealVector &i_c_vars)
  set the inactive continuous variables

• const IntVector & inactive_discrete_variables () const
  return the inactive discrete variables

• void inactive_discrete_variables (const IntVector &i_d_vars)
  set the inactive discrete variables

• const StringArray & inactive_continuous_variable_labels () const
  return the inactive continuous variable labels

• void inactive_continuous_variable_labels (const StringArray &i_c_v_labels)
  set the inactive continuous variable labels

• const StringArray & inactive_discrete_variable_labels () const
  return the inactive discrete variable labels

• void inactive_discrete_variable_labels (const StringArray &i_d_v_labels)
  set the inactive discrete variable labels

• size_t acv () const
  returns total number of continuous vars

• size_t adv () const
  returns total number of discrete vars

• RealVector all_continuous_variables () const
  returns a single array with all continuous variables

• void all_continuous_variables (const RealVector &a_c_vars)
  sets all continuous variables using a single array

• IntVector all_discrete_variables () const
  returns a single array with all discrete variables
8.34 DistinctVariables Class Reference

- **void all_discrete_variables (const IntVector &a_d_vars)**
  
  sets all discrete variables using a single array

- *StringArray all_continuous_variable_labels () const*
  
  returns a single array with all continuous variable labels

- **void all_continuous_variable_labels (const StringArray &a_c_v_labels)**
  
  sets all continuous variable labels using a single array

- *StringArray all_discrete_variable_labels () const*
  
  returns a single array with all discrete variable labels

- **void all_discrete_variable_labels (const StringArray &a_d_v_labels)**
  
  sets all discrete variable labels using a single array

- *StringArray all_variable_labels () const*
  
  returns a single array with all variable labels

- **void read (istream &s)**
  
  read a variables object from an istream

- **void write (ostream &s) const**
  
  write a variables object to an ostream

- **void write_aprepro (ostream &s) const**
  
  write a variables object to an ostream in aprepro format

- **void read.annotated (istream &s)**
  
  read a variables object in annotated format from an istream

- **void write.annotated (ostream &s) const**
  
  write a variables object in annotated format to an ostream

- **void write.tabular (ostream &s) const**
  
  write a variables object in tabular format to an ostream

- **void read (BiStream &s)**
  
  read a variables object from the binary restart stream

- **void write (BoStream &s) const**
  
  write a variables object to the binary restart stream

- **void read (MPIUnpackBuffer &s)**
  
  read a variables object from a packed MPI buffer

- **void write (MPIPackBuffer &s) const**
  
  write a variables object to a packed MPI buffer
Protected Member Functions

- void copy_rep (const Variables *vars_rep)
  Used by copy() to copy the contents of a letter class.

- void reshape_rep (const Sizet2DArray &vars_comps)
  Used by reshape() to reshape the contents of a letter class.

Private Member Functions

- void build_types_ids ()
  construct VarTypes and VarIds arrays using variablesComponents

Private Attributes

- RealVector continuousDesignVars
  the continuous design variables array

- IntVector discreteDesignVars
  the discrete design variables array

- RealVector uncertainVars
  the uncertain variables array

- RealVector continuousStateVars
  the continuous state variables array

- IntVector discreteStateVars
  the discrete state variables array

- StringArray continuousDesignLabels
  the continuous design variables label array

- StringArray discreteDesignLabels
  the discrete design variables label array

- StringArray uncertainLabels
  the uncertain variables label array

- StringArray continuousStateLabels
  the continuous state variables label array

- StringArray discreteStateLabels
  the discrete state variables label array
8.34 DistinctVariables Class Reference

Friends

- bool operator==(const DistinctVariables &vars1, const DistinctVariables &vars2)

8.34.1 Detailed Description

The default data view (no variable or 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 DistinctVariables derived class separates the design, uncertain, and state variable types as well as the continuous and discrete domain types. The result is separate arrays for continuous design, discrete design, uncertain, continuous state, and discrete state variables. This is the default approach, so all iterators and strategies not specifically utilizing the All or Merged views use this approach (see Variables::get_-variables(problem_db)).

8.34.2 Constructor & Destructor Documentation

8.34.2.1 DistinctVariables (const ProblemDescDB & problem_db, const 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. Extract distinct variable types and labels (VariablesUtil is not used).

8.34.3 Friends And Related Function Documentation

8.34.3.1 bool operator==(const DistinctVariables & vars1, const DistinctVariables & vars2) [friend]

equality operator

Checks each array using operator== from data_types.C. Labels are ignored.

The documentation for this class was generated from the following files:

- DistinctVariables.H
- DistinctVariables.C
8.35 DOTOptimizer Class Reference

Wrapper class for the DOT optimization library.

Inheritance diagram for DOTOptimizer:

```
DOTOptimizer
  |  |
  |  |
  |  |
Minimizer
  |  |
  |  |
  |  |
Optimizer
  |  |
  |  |
  |  |
Iterator
```

### Public Member Functions

- **DOTOptimizer (Model &model)**
  * constructor
- **~DOTOptimizer ()**
  * destructor
- **void find_optimum ()**
  * Redefines the run virtual function for the optimizer branch.

### Protected Member Functions

- **virtual void derived_pre_run ()**
  * performs run-time set up

### Private Member Functions

- **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).

- int optimizationType
  MINMAX from DOT manual (minimize or maximize).

- 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
  total number of nonlinear constraints seen by DOT
8.35.1 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.

8.35.2 Member Data Documentation

8.35.2.1 int dotInfo [private]

INFO from DOT manual.
Information requested by DOT: 0=optimization complete, 1=get values, 2=get gradients

8.35.2.2 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).

8.35.2.3 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.

8.35.2.4 int printControl [private]

IPRINT from DOT manual (controls output verbosity).
Values range from 0 (least output) to 7 (most output).

8.35.2.5 int optimizationType [private]

MINMAX from DOT manual (minimize or maximize).
Values of 0 or -1 (minimize) or 1 (maximize).

8.35.2.6 RealArray realCntlParmArray [private]

RPRM from DOT manual.
Array of real control parameters.

8.35.2.7 IntArray intCntlParmArray [private]

IPRM from DOT manual.
Array of integer control parameters.

8.35.2.8 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).

8.35.2.9 SizetList constraintMappingIndices [private]

Response constraints used in computing the DOT constraints.
The length of the list corresponds to the number of DOT constraints, and each entry in the list points to the corresponding DAKOTA constraint.
8.35.2.10  **RealList constraintMappingMultipliers**  [private]

the DOT constraints.
The length of the list corresponds to the number of DOT constraints, and each entry in the list contains a multiplier
for the DAKOTA constraint identified with constraintMappingIndices. These multipliers are currently +1 or -1.

8.35.2.11  **RealList constraintMappingOffsets**  [private]

DOT constraints.
The length of the list corresponds to the number of DOT constraints, and each entry in the list contains an offset
for the DAKOTA constraint identified with constraintMappingIndices. These offsets involve inequality bounds or
equality targets, since DOT assumes constraint allowables = 0.

The documentation for this class was generated from the following files:

- DOTOptimizer.H
- DOTOptimizer.C
8.36 EffGlobalOptimizer Class Reference

Implementation of Efficient Global Optimization algorithm.

Inheritance diagram for EffGlobalOptimizer:

```
  Iterator
     ↓
  Minimizer
     ↓
  Optimizer
     ↓
EffGlobalOptimizer
```

Public Member Functions

- **EffGlobalOptimizer** (Model &model)
  
  *standard constructor*

- **~EffGlobalOptimizer** ()
  
  *alternate constructor for instantiations "on the fly" destructor*

- void **find_optimum** ()
  
  *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 **get_best_sample** ()
  
  *improvement function*

- Real **expected_improvement** (const RealVector &expected_values, const RealVector &c_variables)
  
  *expected improvement function for the GP*

- RealVector **expected_violation** (const RealVector &variances, const RealVector &c_variables)
  
  *expected violation function for the constraint functions*
void update_penalty()
  initialize and update the penaltyParameter

void update_augmented_lagrange_multipliers(const RealVector &fn_vals)
  initialize and update the Lagrange multipliers for augmented Lagrangian

Real augmented_lagrangian_merit(const RealVector &fn_vals, 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

Real objective(const RealVector &fn_vals)
  compute a composite objective value from one or more objective functions

Real constraintViolation(const RealVector &fn_vals, const Real &constraint_tol)
  compute the constraint violation from a set of function values

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)
  Expected Improvement (EIF) problem formulation for PMA.

static void merit_objective_eval(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  defines an augmented Lagrangian merit function from the sub_model_response

Private Attributes

String setUpType
  (user-supplied functions mode for "on the fly" instantiations).

Real Pi
  \( \pi = 3.1415926535897932385 \)

Model meritModel
  recast model to create an augmented Lagrangian merit function

Model fHatModel
  if constraints present

Model eifModel
  recast model used in solving the max(EIF) sub-problem
- **Iterator eifOptimizer**
  iterator used 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

- **RealVector augLagrangeMult**
  Lagrange multipliers for augmented Lagrangian calculations.

- **Real penaltyParameter**
  penalty calculations; increased in `update_penalty()`

- **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**
  Lagrangian updates (refer to Conn, Gould, and Toint, section 14.4).

- **size_t numFns**
  number of response functions

- **size_t numVars**
  number of active continuous variables

- **size_t numObjFns**
  number of objective functions

- **size_t numNonlinIneqConstr**
  number of nonlinear inequality constraints

- **size_t numNonlinEqConstr**
  number of nonlinear equality constraints

- **RealVector nonlinIneqLowerBnds**
nonlinear inequality constraint lower bounds

- RealVector nonlinIneqUpperBnds
  nonlinear inequality constraint upper bounds

- RealVector nonlinEqTargets
  nonlinear equality constraint targets

- Real bigRealBoundSize
  cutoff value for continuous bounds

- size_t approxIters
  iteration counter

Static Private Attributes

- static EffGlobalOptimizer * effGlobalInstance
  functions in order to avoid the need for static data

8.36.1 Detailed Description

Implementation of Efficient Global Optimization algorithm.

The EffGlobalOptimizer class provides an implementation of the Efficient Global Optimization algorithm developed by Jones, Schonlau, & Welch.

The user input mappings are as follows:

8.36.2 Constructor & Destructor Documentation

8.36.2.1 ~EffGlobalOptimizer ()

alternate constructor for instantiations "on the fly" destructor

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

8.36.3 Member Function Documentation
8.36.3.1 Real objective (const RealVector & fn_vals) [private]

compute a composite objective value from one or more objective functions
The composite objective computation sums up the contributions from one or more objective functions using the
multiobjective weights.

8.36.3.2 Real constraint_violation (const RealVector & fn_vals, const Real & constraint_tol) [private]

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).

The documentation for this class was generated from the following files:

- EffGlobalOptimizer.H
- EffGlobalOptimizer.C
8.37  **ErrorTable Struct Reference**

Data structure to hold errors.

**Public Attributes**

- **CtelRegexp::RStatus** rc
  
  *Enumerated type to hold status codes.*

- **const char** *msg*
  
  *Holds character string error message.*

8.37.1  **Detailed Description**

Data structure to hold errors.

This module implements a C++ wrapper for Regular Expressions based on the public domain engine for regular expressions released by: Copyright (c) 1986 by University of Toronto. Written by Henry Spencer. Not derived from licensed software.

The documentation for this struct was generated from the following file:

- **CtelRegExp.C**
8.38  ForkAnalysisCode Class Reference

simulations using forks.

Inheritance diagram for ForkAnalysisCode::

```
AnalysisCode
ForkAnalysisCode
```

Public Member Functions

- **ForkAnalysisCode** (const **ProblemDescDB** &problem_db)
  *constructor*

- **~ForkAnalysisCode** ()
  *destructor*

- **pid_t fork_program** (const bool block_flag)
  *for completion using waitpid() if block_flag is true*

- **void check_status** (const int status)
  *error code was returned*

- **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** (const int analysis_id)
  *set argList for execution of the specified analysis driver*

Private Attributes

- **StringArray argList**
  *These are converted to an array of const char*’s in fork_program().*
8.38.1 Detailed Description

Simulations using forks.

ForkAnalysisCode creates a copy of the parent DAKOTA process using fork()/vfork() and then replaces the copy with a simulation process using execvp(). The parent process can then use waitpid() to wait on completion of the simulation process.

8.38.2 Member Function Documentation

8.38.2.1 void check_status (const int status)

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().

The documentation for this class was generated from the following files:

- ForkAnalysisCode.H
- ForkAnalysisCode.C
8.39 ForkApplicInterface Class Reference

using forks.

Inheritance diagram for ForkApplicInterface::

```
  Interface
   |   |
---+---+---
|   |   |
|   |   |
 ApplicationInterface
   |   |
---+---+---
|   |   |
|   |   |
 ForkApplicInterface
```

Public Member Functions

- **ForkApplicInterface** (const ProblemDescDB &problem_db)
  constructor

- **~ForkApplicInterface** ()
  destructor

- void **derived_map** (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  that is specific to a derived class.

- void **derived_map_asynch** (const ParamResponsePair &pair)
  asynchronous evaluation that is specific to a derived class.

- void **derived_synch** (PRPList &prp_list)
  classes. This version waits for at least one completion.

- void **derived_synch_nowait** (PRPList &prp_list)
  any completions if none are immediately available.

- int **derived_synchronous_local_analysis** (const int &analysis_id)
- const **StringBuilder** & **analysis_drivers** ()
  retrieve the analysis drivers specification for application interfaces

Private Member Functions

- void **derived_synch_kernel** (PRPList &prp_list, const pid_t pid)
### derived_synch_nowait()

- **pid_t fork_application** (const bool block_flag)
  
  _filter, analysis programs, and output filter_

- **void asynchronous_local_analyses** (const int &start, const int &end, const int &step)
  
  _execute analyses asynchronously on the local processor_

- **void synchronous_local_analyses** (const int &start, const int &end, const int &step)
  
  _execute analyses synchronously on the local processor_

- **void serve_analyses_asynch**()
  
  _serve the analysis scheduler and execute analysis jobs asynchronously_

### Private Attributes

- **ForkAnalysisCode forkSimulator**
  
  _individual programs and checking fork exit status_

- **std::map< pid_t, int > processIdMap**
  
  _asynchronous evaluations_

### 8.39.1 Detailed Description

using forks.

**ForkApplicInterface** uses a **ForkAnalysisCode** object for performing simulation invocations.

### 8.39.2 Member Function Documentation

#### 8.39.2.1 int derived_synchronous_local_analysis (const int & analysis_id) [inline, virtual]

This code provides the derived function used by **ApplicationInterface::serve_analyses_synch()** as well as a convenience function for **ForkApplicInterface::synchronous_local_analyses()** below.

Reimplemented from **ApplicationInterface**.

#### 8.39.2.2 pid_t fork_application (const bool block_flag) [private]

_filter, analysis programs, and output filter_
Manage the input filter, 1 or more analysis programs, and the output filter in blocking or nonblocking mode as
governed by block_flag. In the case of a single analysis and no filters, a single fork is performed, while in other
cases, an initial fork is reforked multiple times. Called from derived_map() with block_flag == BLOCK and
from derived_map_asynch() with block_flag == FALL_THROUGH. Uses ForkAnalysisCode::fork_program() to
spawn individual program components within the function evaluation.

8.39.2.3 void asynchronous_local_analyses (const int & start, const int & end, const int & step)
   [private]
execute analyses asynchronously on the local processor
Schedule analyses asynchronously on the local processor using a self-scheduling approach (start to
end in step increments). Concurrency is limited by asynchLocalAnalysisConcurrency. Modeled af-
after ApplicationInterface::asynchronous_local_evaluations(). NOTE: This function should be elevated to
ApplicationInterface if and when another derived interface class supports asynchronous local analyses.

8.39.2.4 void synchronous_local_analyses (const int & start, const int & end, const int & step)
   [inline, private]
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().

8.39.2.5 void serve_analyses_asynch () [private]
serve the analysis scheduler and execute analysis jobs asynchronously
This code runs multiple async 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.
The documentation for this class was generated from the following files:

- ForkApplicInterface.H
- ForkApplicInterface.C
8.40  FSUDesignCompExp Class Reference

Wrapper class for the FSUDace QMC/CVT library.

Inheritance diagram for FSUDesignCompExp:

```
       Iterator
          |
          v
       Analyzer
           |
           v
       PStudDACE
            |
            v
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 extract_trends** ()

  *Redefines the run_iterator virtual function for the PStudy/DACE branch.*

- **void sampling_reset** (int min_samples, bool all_data_flag, bool stats_flag)

  *reset sampling iterator*

- **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** (const 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
  
  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

- bool volQualityFlag
  
  flag which specifies evaluating the volumetric quality measures

- bool varBasedDecompFlag
  
  sensitivity analysis metrics

- IntVector sequenceStart
  
  variable sampled. Default is 0 0 0 (e.g. for three random variables).

- IntVector sequenceLeap
  
  generated. Default is 1 1 1 (e.g. for three random vars.)

- IntVector primeBase
  
  generated. Default is 2 3 5 (e.g., for three random vars.)

- int originalSeed
  
  (allows repeatable results)

- int randomSeed
  
  current seed for the random number generator

- bool varyPattern
  
  multiple executions are repeatable but not identical.
- int `numCVTTrials`
  
  specifies the number of sample points taken at internal CVT iteration

- int `trialType`
  
  halton (1), uniform (0), or random (-1). Default is random.

### 8.40.1 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.

### 8.40.2 Constructor & Destructor Documentation

#### 8.40.2.1 `FSUDesignCompExp (Model & model)`

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

#### 8.40.2.2 `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.

### 8.40.3 Member Function Documentation

#### 8.40.3.1 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.

The documentation for this class was generated from the following files:
- FSUDesignCompExp.H
- FSUDesignCompExp.C
8.41 FunctionCompare Class Template Reference

Public Member Functions

- `FunctionCompare (bool(*)(func)(const T &, void *), void *)`
  - Constructor that defines the pointer to function and search value.

- `bool operator()(T t) const`
  - The operator() must be defined. Calls the function test_fn.

Private Attributes

- `bool(*)(test_fn)(const T &, void *)`
  - Pointer to test function.

- `void *search_val`
  - Holds the value to search for.

8.41.1 Detailed Description

`template<class T> class Dakota::FunctionCompare<T>`

Internal functor to mimic the RW find and index functions using the STL find_if() method. The class holds a pointer to the test function and the search value.

The documentation for this class was generated from the following file:

- DakotaList.H
8.42 GaussProcApproximation Class Reference

Derived approximation class for Gaussian Process implementation.

Inheritance diagram for GaussProcApproximation::

```
Approximation
   GaussProcApproximation
```

Public Member Functions

- **GaussProcApproximation** ()
  *default constructor*

- **GaussProcApproximation** (ProblemDescDB &problem_db, const size_t &num_acv)
  *standard constructor*

- ~**GaussProcApproximation** ()
  *destructor*

Protected Member Functions

- int **num_coefficients** () const
  *derived class approximation type in numVars dimensions*

- int **num_constraints** () const
  *return the number of constraints to be enforced via anchorPoint*

- void **find_coefficients** ()
  *find the covariance parameters governing the Gaussian process response*

- const Real & **get_value** (const RealVector &x)
  *retrieve the function value for a given parameter set x*

- const Real & **get_variance** (const RealVector &x)
  *retrieve the variance of the predicted value for a given parameter set x*
Private Member Functions

- `void GPmodel_build ()`
  Function to compute hyperparameters governing the GP.

- `void GPmodel_apply (const RealVector &new_x, bool variance_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 ()`
  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 ()`
  Error checking

- `void get_process_variance ()`
  The correlation lengthscales

- `void get_cov_matrix ()`
  Calculates the covariance matrix for a given set of input points

- `void get_cov_vector ()`
  Set of inputs upon which the GP is based

- `void optimize_theta_global ()`
  Parameters using NCSUDirect

- `void optimize_theta_multipoint ()`
  Parameters using a gradient-based solver and multiple staring points

- `void predict (bool variance_flag)`
  Calculates the predicted new response value for x in normalized space.

- `Real calc_nll ()`
  Calc_nll (matrix)

- `void calc_grad_nll ()`
  To the correlation lengthscales, theta

- `void run_point_selection ()`
  Estimate the necessary parameters
• void initialize_point_selection ()
  
  initial subset of the training points

• void pointsel_get_errors (RealVector &)
  
  training points and find the errors

• int addpoint (int, IntVector &)
  
  Adds a point to the effective training set. Returns 1 on success.

• int pointsel_add_sel (RealVector &)
  
  them

• Real maxval (RealVector &)
  
  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 ()
  
  likelihood on a grid

• void writex (char[])
  
  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 NEWMAT::ColumnVector &X, NEWMAT::Real &fx, NEWMAT::ColumnVector &grad_x, int &result_mode)
  
  by minimizing the negative log likelihood

• static void constraint_eval (int mode, int n, const NEWMAT::ColumnVector &X, NEWMAT::ColumnVector &g, NEWMAT::Matrix &gradC, int &result_mode)
  
  this function is empty: it is an unconstrained optimization.

• static void negloglikNCSU (int &n, double *x, double &f, int &flag, int *idata, int &isize, double *ddata, int &dsize, char *cdata, int &csize)
  
  static function used by DIRECT to optimize negloglik objective
Private Attributes

- Epetra_SerialDenseMatrix `trainPoints`
  used to create the Gaussian process

- Epetra_SerialDenseMatrix `trainValues`
  An array of response values; one response value per sample site.

- Epetra_SerialDenseVector `trainMeans`
  The mean of the input columns of `trainPoints`.

- Epetra_SerialDenseVector `trainStdvs`
  The standard deviation of the input columns of `trainPoints`.

- Epetra_SerialDenseMatrix `normTrainPoints`
  Current working set of normalized points upon which the GP is based.

- Epetra_SerialDenseMatrix `trendFunction`
  matrix to hold the trend function

- Epetra_SerialDenseMatrix `betaCoeffs`
  matrix to hold the beta coefficients for the trend function

- Epetra_SerialSymDenseMatrix `covMatrix`
  between points Xi and Xj in the initial set of samples

- Epetra_SerialDenseMatrix `covVector`
  between a new point X and point Xj from the initial set of samples

- Epetra_SerialDenseMatrix `approxPoint`
  single point, but it could be generalized to be a vector of points.

- Epetra_SerialDenseMatrix `gradNegLogLikTheta`
  with respect to the theta correlation terms

- Epetra_SerialSpdDenseSolver `covSlvr`
  the covariance matrix

- Epetra_SerialDenseMatrix `normTrainPointsAll`
  Set of all original samples available.

- Epetra_SerialDenseMatrix `trainValuesAll`
  All original samples available.

- Epetra_SerialDenseMatrix `trendFunctionAll`
  Trend function values corresponding to all original samples.
8.42 GaussProcApproximation Class Reference

- size_t numObs
  The number of observations on which the GP surface is built.

- size_t numObsAll
  The original number of observations.

- short trendOrder
  linear, if order = 2, trend is quadratic.

- RealVector thetaParams
  same point. sige is the underlying process error.

- Real procVar
  The process variance, the multiplier of the correlation matrix.

- RealVector pointsAddedIndex
  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

8.42.1 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.

8.42.2 Member Function Documentation
8.42.2.1  void GPmodel_apply (const RealVector & new_x, bool variance_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.
The documentation for this class was generated from the following files:

- GaussProcApproximation.H
- GaussProcApproximation.C
8.43 GenLaguerreOrthogPolynomial Class Reference

Derived orthogonal polynomial class for generalized Laguerre polynomials.
Inheritance diagram for GenLaguerreOrthogPolynomial:

```
OrthogonalPolynomial
   \downarrow
GenLaguerreOrthogPolynomial
```

Public Member Functions

- `GenLaguerreOrthogPolynomial()`
  default constructor

- `GenLaguerreOrthogPolynomial(const Real &alpha_stat)`
  standard constructor

- `~GenLaguerreOrthogPolynomial()`
  destructor

Protected Member Functions

- `const Real &get_value(const Real &x, size_t n)`
  parameter x

- `const Real &get_gradient(const Real &x, size_t n)`
  given parameter x

- `const Real &norm_squared(size_t n)`
  return the inner product \( <L^{\alpha}_n,L^{\alpha}_n> = ||L^{\alpha}_n||^2 \)

- `const RealVector &gauss_points(size_t n)`
  corresponding to polynomial order n

- `const RealVector &gauss_weights(size_t n)`
  corresponding to polynomial order n

- `void alpha_stat(const Real &alpha)`
  set alphaPoly using the conversion alphaPoly = alpha_stat-1.
Private Attributes

- Real alphaPoly

by Abramowitz and Stegun (differs from statistical PDF notation)

8.43.1 Detailed Description

Derived orthogonal polynomial class for generalized Laguerre polynomials.

The GenLaguerreOrthogPolynomial class evaluates a univariate generalized/associated Laguerre polynomial \( L^\alpha_n \) of a particular order. These polynomials are orthogonal with respect to the weight function \( x^\alpha \exp(-x) \) when integrated over the support range of \([0, +\infty)\). This corresponds to the probability density function \( f(x) = x^\alpha \exp(-x) / \Gamma(\alpha+1) \) for the standard gamma distribution, although common statistical PDF parameter conventions (see, e.g., the uncertain variables section in the DAKOTA Reference Manual) and the Abramowitz and Stegun orthogonal polynomial parameter conventions require an offset conversion in this case (\( \alpha_{poly} = \alpha_{stat} - 1 \) with the poly definition used in both cases above). It enables (mixed) multidimensional orthogonal polynomial basis functions within OrthogPolyApproximation. A special case is the LaguerreOrthogPolynomial (implemented separately), for which \( \alpha_{poly} = 0 \) and weight function = \( \exp(-x) \) (the standard exponential distribution).

The documentation for this class was generated from the following files:

- GenLaguerreOrthogPolynomial.H
- GenLaguerreOrthogPolynomial.C
8.44 GetLongOpt Class Reference

(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 (ostream &outfile=cout) const

  *Print usage information to outfile.*

- void usage (const char *str)

  *Change header of usage output to str.*
Private Member Functions

- char * basename (char *const p) const
  
take 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
  
lst entry in option table

8.44.1 Detailed Description

(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 "=".

8.44.2 Constructor & Destructor Documentation
8.44 GetLongOpt Class Reference

8.44.21  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.

8.44.3 Member Function Documentation

8.44.3.1  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.

8.44.3.2  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.

8.44.3.3  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.

8.44.3.4  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.
8.44.3.5  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.

The documentation for this class was generated from the following files:

- CommandLineHandler.H
- CommandLineHandler.C
8.45 Graphics Class Reference

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 String &tabular_data_file)**
  
  *opens the tabular data file stream and prints the headings*

- **void add_datapoint (const Variables &vars, const Response &response)**
  
  *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)**
  
  *for a single window in the 2d graphics*

- **void show_data_3d (const RealVector &X, const RealVector &Y, const RealMatrix &F)**
  
  *generate a new 3d plot for F(X,Y)*

- **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 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 win3dOn
  flag to indicate if 3D 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

• String tabularCntrLabel
  label for counter used in first line comment w/i the tabular data file

• ofstream tabularDataFStream
  file stream for tabulation of graphics data within compute_response

8.45.1 Detailed Description

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).
8.45.2 Member Function Documentation

8.45.2.1 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).

8.45.2.2 void create_tabular_datastream (const Variables & vars, const Response & response, const 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.

8.45.2.3 void add_datapoint (const Variables & vars, const Response & response)
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.

8.45.2.4 void add_datapoint (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.

8.45.2.5 void new_dataset (int i)
for a single window in the 2d graphics
Used for displaying multiple data sets within the same plot.

8.45.2.6 void show_data_3d (const RealVector & X, const RealVector & Y, const RealMatrix & F)
generate a new 3d plot for F(X,Y)
3D plotting clears data set and builds from scratch each time show_data3d is called. This still involves an event loop waiting for a mouse click (right button) to continue. X = 1-D x grid values only and Y = 1-D Y grid values only [X and Y are _not_ (X,Y) pairs]. F = 2-d grid of values for a single function for all (X,Y) combinations.

The documentation for this class was generated from the following files:
- DakotaGraphics.H
- DakotaGraphics.C
8.46 GridApplicInterface Class Reference

using grid services such as Condor or Globus.
Inheritance diagram for GridApplicInterface::

```
  Interface
  |      |
  v      v
ApplicationInterface
  |      |
  v      v
GridApplicInterface
```

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)
  *that is specific to a derived class.*

- void **derived_map_asynch** (const ParamResponsePair &pair)
  *asynchronous evaluation that is specific to a derived class.*

- void **derived_synch** (PRPList &prp_list)
  *classes. This version waits for at least one completion.*

- void **derived_synch_nowait** (PRPList &prp_list)
  *any completions if none are immediately available.*

- int **derived_synchronous_local_analysis** (const int &analysis_id)

Public Attributes

- **SysCallAnalysisCode** code
  *Used to read/write parameter files and responses.*
Protected Member Functions

- void derived_synch_kernel (PRPList &prp_list)
  Convenience function for common code between wait and nowait case.

- bool grid_file_test (const String &root_file)
  test file(s) for existence based on root_file name

Protected Attributes

- IntSet idSet
  system call evaluations

- IntShortMap failCountMap
  map linking function evaluation id's to number of response read failures

- start_grid_computing_t start_grid_computing
  handle to dynamically linked start_grid_computing function

- perform_analysis_t perform_analysis
  handle to dynamically linked perform_analysis grid function

- get_jobs_completed_t get_jobs_completed
  handle to dynamically linked get_jobs_completed grid function

- stop_grid_computing_t stop_grid_computing
  handle to dynamically linked stop_grid_computing function

8.46.1 Detailed Description

using grid services such as Condor or Globus.
This class is currently a modified copy of SysCallApplicInterface adapted for use with an external grid dervices library which was dynamically linked using dlopen() services.

8.46.2 Member Function Documentation

8.46.2.1 int derived_synchronous_local_analysis (const int & analysis_id) [inline, virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().
TODO - allow local analyses?????
Reimplemented from ApplicationInterface.

The documentation for this class was generated from the following files:

- GridApplicInterface.H
- GridApplicInterface.C
8.47 HermiteOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Hermite polynomials.

Inheritance diagram for HermiteOrthogPolynomial::

```
OrthogonalPolynomial
  
HermiteOrthogPolynomial
```

Public Member Functions

- **HermiteOrthogPolynomial ()**
  *default constructor*

- **~HermiteOrthogPolynomial ()**
  *destructor*

Protected Member Functions

- **const Real & get_value (const Real &x, size_t n)**
  *retrieve the Hermite polynomial value for a given parameter x*

- **const Real & get_gradient (const Real &x, size_t n)**
  *retrieve the Hermite polynomial gradient for a given parameter x*

- **const Real & norm_squared (size_t n)**
  *return the inner product $<He_n,He_n> = ||He_n||^2$*

- **const RealVector & gauss_points (size_t n)**
  *polynomial order n*

- **const RealVector & gauss_weights (size_t n)**
  *polynomial order n*
8.47 HermiteOrthogPolynomial Class Reference

8.47.1 Detailed Description

Derived orthogonal polynomial class for Hermite polynomials.

The **HermiteOrthogPolynomial** class evaluates a univariate Hermite polynomial of a particular order. It uses the "probabilist’s" formulation for which the polynomials are orthogonal with respect to the weight function $1/\sqrt{2\pi}$ $\exp(-x^2/2)$ when integrated over the support range of $[-\infty, +\infty]$. It enables (mixed) multi-dimensional orthogonal polynomial basis functions within **OrthogPolyApproximation**.

The documentation for this class was generated from the following files:

- HermiteOrthogPolynomial.H
- HermiteOrthogPolynomial.C
8.48 **HierarchSurrModel Class Reference**

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 **ResponseArray & 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 surrogate_bypass (bool bypass_flag)

for any lower-level surrogates.

void surrogate_function_indices (const IntSet &surr_fn_indices)

(re)set the surrogate index set in SurrogateModel::surrogateFnIndices

void build_approximation ()

correction of lowFidelityModel results

void component_parallel_mode (short mode)

lowFidelityModel and highFidelityModel

void derived_init_communicators (const 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 (const int &max_iterator_concurrency, bool recurse_flag=true)

highFidelityModel

void derived_free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)

(request forwarded to lowFidelityModel and highFidelityModel)

void serve ()

stop_servers().

void stop_servers ()

HierarchSurrModel is complete.

int evaluation_id () const

Return the current evaluation id for the HierarchSurrModel.

void set_evaluation_reference ()

(request forwarded to lowFidelityModel and highFidelityModel)

void print_evaluation_summary (ostream &s, bool minimal_header=false, bool relative_count=true) const

(request forwarded to lowFidelityModel and highFidelityModel)
Private Member Functions

- void update_model(Model &model)
  
  with current variable values/bounds/labels

Private Attributes

- int hierModelEvals
  
  derived_asynch_compute_response()

- IntResponseMap cachedTruthRespMap
  
  portions were still pending.

- Model lowFidelityModel
  
  a data fit surrogate on a low fidelity model).

- Model highFidelityModel
  
  fidelity results. Model is of arbitrary type and supports recursions.

- Response highFidRefResponse
  
  and used for calculating corrections.

8.48.1 Detailed Description

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.

8.48.2 Member Function Documentation

8.48.2.1 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.
8.48 HierarchSurrModel Class Reference

8.48.2.2  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.

8.48.2.3  const ResponseArray & derived_synchronize () [protected, virtual]

portion of synchronize() specific to HierarchSurrModel

Blocking retrieval of asynchronous evaluations from lowFidelityModel, 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.

8.48.2.4  const IntResponseMap & derived_synchronize_nowait () [protected, virtual]

portion of synchronize_nowait() specific to HierarchSurrModel

Nonblocking retrieval of asynchronous evaluations 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.

8.48.2.5  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.

The documentation for this class was generated from the following files:

- HierarchSurrModel.H
- HierarchSurrModel.C
8.49 IDRProblemDescDB Class Reference

The derived input file database utilizing the IDR parser.

Inheritance diagram for IDRProblemDescDB:

```
ProblemDescDB
    IDRProblemDescDB
```

**Public Member Functions**

- IDRProblemDescDB (ParallelLibrary &parallel_lib)
  
  *constructor*

- ~IDRProblemDescDB ()
  
  *destructor*

- void derived_manage_inputs (const char *dakota_input_file)
  
  *database using IDR.*

**Static Public Member Functions**

- static void strategy_kwhandler (const struct FunctionData *parsed_data)
  
  *specification is parsed*

- static void method_kwhandler (const struct FunctionData *parsed_data)
  
  *specification is parsed*

- static void model_kwhandler (const struct FunctionData *parsed_data)
  
  *specification is parsed*

- static void variables_kwhandler (const struct FunctionData *parsed_data)
  
  *variables specification is parsed*

- static void interface_kwhandler (const struct FunctionData *parsed_data)
  
  *interface specification is parsed*

- static void responses_kwhandler (const struct FunctionData *parsed_data)
  
  *responses specification is parsed*
Static Private Member Functions

- static void idr_kw_id_error (const char *kw)
  
  *Error handler for missing required IDR keyword.*

- static Int idr_find_id (Int *id_pos, const Int cntr, const char *id, const char **id_list, const char *kw)
  
  *instances of a particular keyword*

- static Int ** idr_get_int_table (const struct FunctionData *parsed_data, Int identifier, Int &table_len, Int num_lists, Int list_entry_len)
  
  *Function for creating an IDR table of Ints.*

- static Real ** idr_get_real_table (const struct FunctionData *parsed_data, Int identifier, Int &table_len, Int num_lists, Int list_entry_len)
  
  *Function for creating an IDR table of Reals.*

- static char *** idr_get_string_table (const struct FunctionData *parsed_data, Int identifier, Int &table_len, Int num_lists, Int list_entry_len)
  
  *Function for creating an IDR table of strings.*

Static Private Attributes

- static IDRProblemDescDB * pDDBInstance
  
  *functions in order to avoid the need for static data*

- static Int ** intTable
  
  *integer table populated in idr_get_int_table()*

- static Real ** realTable
  
  *real table populated in idr_get_real_table()*

- static char *** stringTable
  
  *string table populated in idr_get_string_table()*

8.49.1 Detailed Description

The derived input file database utilizing the IDR parser.

The IDRProblemDescDB class is a database for DAKOTA input file data that is populated by the Input Deck Reader (IDR) parser. When the parser reads a complete keyword (delimited by a newline), it calls the corresponding kwhandler function from this class which populates the corresponding Data object from the base class. For information on modifying the IDR input parsing procedures, refer to Dakota/docs/Dev_Spec_Change.dox
8.49.2 Member Function Documentation

8.49.2.1 void derived_manage_inputs (const char *dakota_input_file) [virtual]

database using IDR.

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.

The documentation for this class was generated from the following files:

- IDRProblemDescDB.H
- IDRProblemDescDB.C
8.50 Interface Class Reference

Base class for the interface class hierarchy.

Inheritance diagram for Interface::

```
  Interface
  |   |
  v   v
ApplicationInterface  ApproximationInterface
  |
DirectFnApplicInterface
  |
  ForkApplicInterface
  |
  GridApplicInterface
  |
  SysCallApplicInterface
```

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, const bool asynch_-flag=false)
  
  variables to the responses.

- virtual const ResponseArray & synch ()
  
  recovers data from a series of asynchronous evaluations (blocking)

- virtual const RespResponseMap & synch_nowait ()
  
  recovers data from a series of asynchronous evaluations (nonblocking)
virtual void serve_evaluations ()
    evaluation server function for multiprocessor executions

virtual void stop_evaluation_servers ()
    send messages from iterator rank 0 to terminate evaluation servers

virtual void init_communicators (const IntArray &message_lengths, const int &max_iterator_concurrency)
    iterator and concurrent multiprocessor analyses within an evaluation.

virtual void set_communicators (const IntArray &message_lengths)
    (the partitions are already allocated in ParallelLibrary).

virtual void free_communicators ()
    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_samples (bool constraint_flag) const
    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 Response &response)
    updates the anchor point for an approximation

virtual void update_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array)
    updates the current data points for an approximation

virtual void append_approximation (const Variables &vars, const Response &response)
    appends a single point to an existing approximation

virtual void append_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array)
    appends multiple points to an existing approximation

virtual void build_approximation (const RealVector &lower_bnds, const RealVector &upper_bnds)
virtual void clear_current ()

clears current data from an approximation interface

virtual void clear_all ()

clears all data from an approximation interface

virtual bool anchor () const

queries the presence of an anchorPoint within an approximation interface

virtual Array< Approximation > & approximations ()

retrieve the Approximations within an ApproximationInterface

virtual const RealVectorArray & approximation_coefficients () const

within an ApproximationInterface

virtual void approximation_coefficients (const RealVectorArray &approx_coeffs) const

within an ApproximationInterface

virtual void print_coefficients (ostream &s, size_t index) const

Approximation instance within an ApproximationInterface.

virtual const RealVector & approximation_variances (const RealVector &c_variables) const

within an ApproximationInterface

virtual const List< SurrogateDataPoint > & approximation_data (size_t index) const

within an ApproximationInterface

virtual const StringArray & analysis_drivers () const

retrieve the analysis drivers specification for application interfaces

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 current function evaluation id for the interface

void set_eval_reference ()

set evaluation count reference points for the interface
void print_eval_summary (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

iterator-evaluation scheduling level

bool is_null () const

function to check interfaceRep (does this envelope contain a letter?)

Protected Member Functions

Interface (BaseConstructor, const ProblemDescDB &problem_db)

derived class constructors - Coplien, p. 139)

Interface (NoDBBaseConstructor)

(NoDBBaseConstructor used for on the fly instantiations without a DB)

void asv_mapping (const ActiveSet &total_set, ActiveSet &algebraic_set, ActiveSet &core_set, const Variables &vars, const Response &response)

from the total Interface evaluation requirements (total_set). Also

void algebraic_mappings (const Variables &vars, const ActiveSet &algebraic_set, Response &algebraic_response)

and the data extracted from the algebraic_mappings file

void response_mapping (const Response &algebraic_response, const Response &core_response, Response &total_response)

from derived_map() to create the total response

Protected Attributes

String interfaceType

the interface type: system, fork, direct, grid, or approximation

String idInterface

(used in print_eval_summary())

bool algebraicMappings

Interface’s parameter to response mapping that is explicit and algebraic.
• bool coreMappings
  
  ApplicationInterface or functionSurfaces for ApproximationInterface.

• int fnEvalId
  
  total interface evaluation counter

• int newFnEvalId
  
  new (non-duplicate) interface evaluation counter

• int fnEvalIdRefPt
  
  iteration reference point for fnEvalId

• int newFnEvalIdRefPt
  
  iteration reference point for newFnEvalId

• 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

- ResponseArray rawResponseArray
  asynchronous evaluations.

- IntResponseMap rawResponseMap
  asynchronous evaluations.

- StringArray responseTags
  (used in print_eval_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

- bool silentFlag
  flag for really quiet (silent) interface output

- bool quietFlag
  flag for quiet interface output

- bool verboseFlag
  flag for verbose interface output

- bool debugFlag
  flag for really verbose (debug) interface output

Private Member Functions

- Interface * get_interface (ProblemDescDB &problem_db)
  Used by the envelope to instantiate the correct letter class.

- int algebraic_function_type (String)
  evaluation call to make
Private Attributes

- `StringArray algebraicVarTags`
  
  set of variable tags from AMPL stub.col

- `SizetArray algebraicACVIndices`
  
  continuous variables

- `SizetArray algebraicDerivIndices`
  
  derivative variables

- `StringArray algebraicFnTags`
  
  set of function tags from AMPL stub.row

- `IntArray algebraicFnTypes`
  
  AMPL objval (conival) calls.

- `RealArray algebraicConstraintWeights`
  
  set of weights for computing Hessian matrices for algebraic constraints;

- `SizetArray algebraicFnIndices`
  
  DAKOTA response functions.

- `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

8.50.1 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.
8.50.2 Constructor & Destructor Documentation

8.50.2.1 Interface ()

default constructor
used in Model envelope class instantiations

8.50.2.2 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.

8.50.2.3 Interface (const Interface & interface)

copy constructor
Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.

8.50.2.4 ~Interface () [virtual]

destructor
Destructor decrements referenceCount and only deletes interfaceRep if referenceCount is zero.

8.50.2.5 Interface (BaseConstructor, const ProblemDescDB & problem_db) [protected]

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).

8.50.3 Member Function Documentation

8.50.3.1 Interface operator= (const Interface & interface)

assignment operator

### 8.50.3.2 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).

### 8.50.3.3 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.

### 8.50.4 Member Data Documentation

#### 8.50.4.1 ResponseArray rawResponseArray [protected]

asynchronous evaluations.

The array is the raw set of responses corresponding to all asynchronous map calls. This raw array is postprocessed (i.e., finite difference gradients merged) in Model::synchronize() where it becomes responseArray.

#### 8.50.4.2 IntResponseMap rawResponseMap [protected]

asynchronous evaluations.

The map is a partial set of completions which are identified through their fn_eval_id key. Postprocessing from raw to combined form (i.e., finite difference gradient merging) is not currently supported in Model::synchronize_nowait().

The documentation for this class was generated from the following files:
- DakotaInterface.H
- DakotaInterface.C
8.51 Iterator Class Reference

Base class for the iterator class hierarchy.

Inheritance diagram for Iterator:

```
Iterator
|___Analyzer
|   |___Minimizer
|   |   |___NonD
|   |   |   |___NonDACE
|   |   |___FitNonD
|   |   |___LeastSq
|   |   |___Optimizer
|   |   |   |___NonDEvidence
|   |   |___NonDIntegration
|   |   |___NonDPolynomialChaos
|   |   |___NonDReliability
|   |   |___NonDSampling
|   |   |___DDACEDesignCompExp
|   |   |___FSUDesignCompExp
|   |   |___EffGlobalOptimizer
|   |___OptDesign
|   |   |___ParamStudy
|   |   |___PSUADEDesignCompExp
|   |   |___EffGlobalOptimizer
|   |___OptDesign
|___Minimizer
|   |___Minimizer
|   |   |___Minimizer
|   |   |___Minimizer
|   |   |___Minimizer
|   |___Minimizer
|___Minimizer
```

Public Member Functions

- **Iterator ()**
  
  *default constructor*

- **Iterator (Model &model)**
  
  *standard constructor for envelope*

- **Iterator (const Iterator &iterator)**
  
  *copy constructor*

- virtual ~Iterator ()
  
  *destructor*

- **Iterator operator= (const Iterator &iterator)**
  
  *assignment operator*

- virtual void run ()
  
  *run the iterator; portion of run_iterator()*

- 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 bool accepts_multiple_points() const
  return is false. Override to return true if appropriate.

- virtual bool returns_multiple_points() const
  return is false. Override to return true if appropriate.

- virtual const VariablesArray & variables_array_results() const
  only be used if returns_multiple_points() returns true.

- virtual const ResponseArray & response_array_results() const
  only be used if returns_multiple_points() returns true.

- virtual void initial_points(const VariablesArray & pts)
  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 print_results(ostream & s) const
  print the final iterator results

- virtual void multi_objective_weights(const RealVector & multi_obj_wts)
  Used by ConcurrentStrategy for Pareto set optimization.

- virtual void sampling_reset(int min_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator

- virtual const String & sampling_scheme() const
  return sampling name

- 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() const
  return the complete set of evaluated variables

- virtual const ResponseArray & all_responses() const
  return the complete set of computed responses

- void pre_run(ostream & s)
  utility function to verbosely perform common operations prior to run()
- void **pre_run** ()
  
  utility function to quietly perform common operations prior to **run**()

- void **run_iterator** (ostream &s)
  
  utility function to automate **pre_run**()/**run**()/**post_run**() verbosely

- void **run_iterator** ()
  
  utility function to automate **pre_run**()/**run**()/**post_run**() quietly

- void **post_run** (ostream &s)
  
  utility function to verbosely perform common operations following **run**()

- void **post_run** ()
  
  utility function to quietly perform common operations following **run**()

- void **assign_rep** (Iterator *iterator_rep, bool ref_count_incr=true)
  
  replaces existing letter with a new one

- const **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 (idMethod)

- const **int** & **maximum_concurrency** () const
  
  return the maximum concurrency supported by the iterator

- void **maximum_concurrency** (const **int** &max_conc)
  
  set the maximum concurrency supported by the iterator

- void **active_set** (const **ActiveSet** &set)
  
  employ evaluate_parameter_sets()

- const **ActiveSet** & **active_set** () const
  
  employ evaluate_parameter_sets()

- void **sub_iterator_flag** (bool si_flag)
  
  set subIteratorFlag

- void **variable_mappings** (const **SizetArray** &e_index1, const **SizetArray** &d_index1, const **SizetArray** &index2)
  
  set primaryCVarMapIndices, primaryDVarMapIndices, secondaryVarMapIndices
• bool is_null () const
  
  function to check iteratorRep (does this envelope contain a letter?)

• Iterator * iterator_rep () const
  
  that are not mapped to the top Iterator level

Protected Member Functions

• Iterator (BaseConstructor, Model &model)
  
  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 void derived_pre_run ()
  
  portions of pre_run specific to derived iterators

• virtual void derived_post_run ()
  
  portions of post_run specific to derived iterators

• virtual const VariablesArray & initial_points () const
  
  be meaningful after a call to initial_points mutator.

Protected Attributes

• Model iteratedModel
  
  or a thin RecastModel wrapped around it

• const 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
8.51 Iterator Class Reference

- 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 `numDiscreteVars`
  number of active discrete vars.

- `ActiveSet activeSet`
  tracks the response data requirements on each function evaluation

- bool `subIteratorFlag`
  (NestedModel::subIterator or DataFitSurrModel::daceIterator)

- `SizetArray primaryCVarMapIndices`
  level iteration

- `SizetArray primaryDVarMapIndices`
  level iteration

- `SizetArray secondaryVarMapIndices`
  "secondary" variable mappings 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
- Real \texttt{fdHessByGradStepSize}
  \textit{using first-order differences of gradients}

- Real \texttt{fdHessByFnStepSize}
  \textit{using second-order differences of function values}

- bool \texttt{silentOutput}
  \textit{flag for really quiet (silent) algorithm output}

- bool \texttt{quietOutput}
  \textit{flag for quiet algorithm output}

- bool \texttt{verboseOutput}
  \textit{flag for verbose algorithm output}

- bool \texttt{debugOutput}
  \textit{flag for really verbose (debug) algorithm output}

- bool \texttt{asynchFlag}
  \textit{copy of the model's asynchronous evaluation flag}

**Private Member Functions**

- \texttt{Iterator * get_iterator(Model &model)}
  \textit{Used by the envelope to instantiate the correct letter class.}

**Private Attributes**

- \texttt{String idMethod}
  \textit{method identifier string from the input file}

- \texttt{Iterator * iteratorRep}
  \textit{pointer to the letter (initialized only for the envelope)}

- \texttt{int referenceCount}
  \textit{number of objects sharing iteratorRep}

**8.51.1 Detailed Description**

Base class for the iterator class hierarchy.

The \texttt{Iterator} class is the base class for one of the primary class hierarchies in DAKOTA. The iterator hierarchy contains all of the iterative algorithms which use repeated execution of simulations as function evaluations. For
memory efficiency and enhanced polymorphism, the iterator hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Iterator) serves as the envelope and one of the derived classes (selected in Iterator::get_iterator()) serves as the letter.

### 8.51.2 Constructor & Destructor Documentation

#### 8.51.2.1 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.

#### 8.51.2.2 Iterator (Model & model)

standard constructor for envelope

Used in iterator instantiations within strategy constructors. Envelope constructor only needs to extract enough data to properly execute get_iterator, since Iterator(BaseConstructor, model) builds the actual base class data inherited by the derived iterators.

#### 8.51.2.3 Iterator (const Iterator & iterator)

copy constructor

Copy constructor manages sharing of iteratorRep and incrementing of referenceCount.

#### 8.51.2.4 ~Iterator () [virtual]

destructor

Destructor decrements referenceCount and only deletes iteratorRep when referenceCount reaches zero.

#### 8.51.2.5 Iterator (BaseConstructor, Model & model) [protected]

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).
8.51.2.6  **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.

8.51.2.7  **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.

8.51.3  **Member Function Documentation**

8.51.3.1  **Iterator operator= (const Iterator & iterator)**

assignment operator

8.51.3.2  **void run ()**  [virtual]

run the iterator; portion of run_iterator()  
**Iterator** supports a construct/pre-run/run/post-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented in LeastSq, NonD, Optimizer, and PStudyDACE.

8.51.3.3  **void print_results (ostream & s) const**  [virtual]

print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in post_run().
Reimplemented in LeastSq, Optimizer, PStudyDACE, NonDEvidence, NonDGlobalReliability, NonDIncremLHSSampling, NonDLHSSampling, NonDLocalReliability, and NonDPolynomialChaos.

8.51.3.4  **void pre_run (ostream & s)**

utility function to verbosely perform common operations prior to run()
8.51 Iterator Class Reference

Iterator supports a construct/pre-run/run/post-run/destruct progression. This function is one form of the overloaded pre-run function. This form accepts an ostream and executes verbosely. It is used for standard stand-alone iterator executions. This function is not virtual: derived portions are defined in derived_pre_run().

8.51.3.5 void pre_run ()

utility function to quietly perform common operations prior to run()

Iterator supports a construct/pre-run/run/post-run/destruct progression. This function is one form of the overloaded pre-run function. This form does not accept an ostream and executes quietly. It is commonly used in sub-iterator executions. This function is not virtual: derived portions are defined in derived_pre_run().

8.51.3.6 void run_iterator (ostream & s)

utility function to automate pre_run()/run()/post_run() verbosely

Iterator supports a construct/pre-run/run/post-run/destruct progression. This non-virtual function is one form of the overloaded run_iterator function which automates the pre-run/run/post-run portions of the progression. This form accepts an ostream and executes verbosely.

8.51.3.7 void run_iterator ()

utility function to automate pre_run()/run()/post_run() quietly

Iterator supports a construct/pre-run/run/post-run/destruct progression. This non-virtual function is one form of the overloaded run_iterator function which automates the pre-run/run/post-run portions of the progression. This form does not accept an ostream and executes quietly.

8.51.3.8 void post_run (ostream & s)

utility function to verbosely perform common operations following run()

Iterator supports a construct/pre-run/run/post-run/destruct progression. This function is one form of the overloaded post-run function. This form accepts an ostream and executes verbosely. This function is not virtual: derived portions are defined in derived_post_run().

8.51.3.9 void post_run ()

utility function to quietly perform common operations following run()

Iterator supports a construct/pre-run/run/post-run/destruct progression. This function is one form of the overloaded post-run function. This form does not accept an ostream and executes quietly. This function is not virtual: derived portions are defined in derived_post_run().

8.51.3.10 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).

8.51.3.11 void derived_pre_run () [protected, virtual]

portions of pre_run specific to derived iterators

Iterator supports a construct/pre-run/run/post-run/destruct progression. This function is the virtual derived class portion of pre_run(). Redefinition by derived classes is optional.

Reimplemented in CONMINOptimizer, DOTOptimizer, NLPQLPOptimizer, SNLLEastSq, and SNLLOptimizer.

8.51.3.12 void derived_post_run () [protected, virtual]

portions of post_run specific to derived iterators

Iterator supports a construct/pre-run/run/post-run/destruct progression. This function is the virtual derived class portion of post_run(). Redefinition by derived classes is optional.

Reimplemented in LeastSq, Optimizer, and SNLLEastSq.

8.51.3.13 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.

8.51.4 Member Data Documentation
8.51 Iterator Class Reference

8.51.4.1 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.

8.51.4.2 Real fdHessByGradStepSize [protected]
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.

8.51.4.3 Real fdHessByFnStepSize [protected]
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.
The documentation for this class was generated from the following files:

- DakotaIterator.H
- DakotaIterator.C
8.52 JacobiOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Jacobi polynomials.

Inheritance diagram for JacobiOrthogPolynomial::

[Diagram showing inheritance structure]

Public Member Functions

- JacobiOrthogPolynomial ()
  
  *default constructor*

- JacobiOrthogPolynomial (const Real &alpha_stat, const Real &beta_stat)
  
  *standard constructor*

- ~JacobiOrthogPolynomial ()
  
  *destructor*

Protected Member Functions

- const Real & get_value (const Real &x, size_t n)
  
  *retrieve the Jacobi polynomial value for a given parameter x*

- const Real & get_gradient (const Real &x, size_t n)
  
  *retrieve the Jacobi polynomial gradient for a given parameter x*

- const Real & norm_squared (size_t n)
  
  \[ \|P_n^{(\alpha,\beta)}\|^2 \]

- const RealVector & gauss_points (size_t n)
  
  *polynomial order n*

- const RealVector & gauss_weights (size_t n)
  
  *polynomial order n*

- void alpha_stat (const Real &alpha)
set betaPoly using the conversion betaPoly = alpha_stat - 1.

- void beta_stat (const Real &beta)
  set alphaPoly using the conversion alphaPoly = beta_stat - 1.

Private Attributes

- Real alphaPoly
  Abramowitz and Stegun (differs from statistical PDF notation).

- Real betaPoly
  Abramowitz and Stegun (differs from statistical PDF notation).

8.52.1 Detailed Description

Derived orthogonal polynomial class for Jacobi polynomials.

The JacobiOrthogPolynomial class evaluates a univariate Jacobi polynomial $P^{\alpha}(\alpha,\beta)_n$ of a particular order. These polynomials are orthogonal with respect to the weight function $(1-x)^\alpha (1+x)^\beta$ when integrated over the support range of \([-1,+1]\). This corresponds to the probability density function $f(x) = (1-x)^\alpha (1+x)^\beta / (2^{\alpha+\beta+1} B(\alpha+1,\beta+1))$ for the beta distribution for \([L,U]=[-1,1]\), where common statistical PDF notation conventions (see, e.g., the uncertain variables section in the DAKOTA Reference Manual) and the Abramowiz and Stegun orthogonal polynomial conventions are inverted and require conversion in this case (alpha_poly = beta_stat - 1; beta_poly = alpha_stat - 1 with the poly definitions used in both cases above). It enables (mixed) multidimensional orthogonal polynomial basis functions within OrthogPolyApproximation. A special case is the LegendreOrthogPolynomial (implemented separately), for which alpha_poly = beta_poly = 0.

The documentation for this class was generated from the following files:

- JacobiOrthogPolynomial.H
- JacobiOrthogPolynomial.C
8.53 JEGAOptimizer Class Reference

A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

Inheritance diagram for JEGAOptimizer::

```
Iterator
 |
 V
 Minimizer
 |
 V
 Optimizer
 |
 JEGAOptimizer
```

**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.

- const JEGA::Utilities::Design * GetBestSolution (const JEGA::Utilities::DesignOFSortSet &from)
  Chooses the best Design from a set of solutions taking into account the algorithm type.

- const JEGA::Utilities::Design * GetBestMOSolution (const JEGA::Utilities::DesignOFSortSet &from)
  Chooses the best Design from a set of solutions assuming that they are generated by a multi objective algorithm.

- const JEGA::Utilities::Design * GetBestSOSolution (const JEGA::Utilities::DesignOFSortSet &from)
  Chooses the best Design 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.

- void resize_variables_results_array (std::size_t newsize)
  Safely resizes the best variables array taking into account the requirements put forth by the envelope-letter design pattern.

- void resize_response_results_array (std::size_t newsize)
  Safely resizes the best response array taking into account the requirements put forth by the envelope-letter design pattern.
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 _sogaMethodText**

  The text that indicates the SOGA method.

- **static const std::string _mogaMethodText**

  The text that indicates the MOGA method.

Classes

- **class Driver**

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

- **class EvaluatorCreator**

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

8.53.1 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).

8.53.2 Constructor & Destructor Documentation
8.53 JEGAOptimizer Class Reference

8.53.2.1 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.

8.53.3 Member Function Documentation

8.53.3.1 void LoadDakotaResponses (const JEGA::Utilities::Design & from, Variables & vars, Response & resp) const [protected]

Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects.

This version is meant for the case where a Variables and a Response object exist and just need to be loaded.

Parameters:

from The JEGA Design class object from which to extract the variable and response information for Dakota.

vars The Dakota::Variables object into which to load the design variable values of from.

resp The Dakota::Response object into which to load the objective function and constraint values of from.

8.53.3.2 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.

8.53.3.3 void LoadAlgorithmConfig (JEGA::FrontEnd::AlgorithmConfig & aConfig) [protected]

Completely initializes the supplied algorithm configuration.

This loads the supplied configuration object with appropriate data retrieved from the parameter database.

Parameters:

aConfig The algorithm configuration object to load.
8.53.3.4 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.

8.53.3.5 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.

8.53.3.6 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.

8.53.3.7 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.
8.53.3.8 const JEGA::Utilities::Design* GetBestSolution (const JEGA::Utilities::DesignOFSortSet & from) [protected]

Chooses the best Design from a set of solutions taking into account the algorithm type. Eventually this functionality must be moved into a separate post-processing application for MO datasets.

8.53.3.9 const JEGA::Utilities::Design* GetBestMOSolution (const JEGA::Utilities::DesignOFSortSet & from) [protected]

Chooses the best Design 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.

8.53.3.10 const JEGA::Utilities::Design* GetBestSOSolution (const JEGA::Utilities::DesignOFSortSet & from) [protected]

Chooses the best Design 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.

8.53.3.11 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.

8.53.3.12 void resize_variables_results_array (std::size_t newsize) [protected]

Safely resizes the best variables array taking into account the requirements put forth by the envelope-letter design pattern. Do not directly call resize on the bestVariablesArray object unless you intend to share the internal content (letter) with other objects after assignment.
Parameters:

$newsize$ The new size for the variables array.

8.53.3.13 void resize_response_results_array (std::size_t $newsize$) [protected]

Safely resizes the best response array taking into account the requirements put forth by the envelope-letter design pattern.

Do not directly call resize on the bestResponseArray object unless you intend to share the internal content (letter) with other objects after assignment.

Parameters:

$newsize$ The new size for the responses array.

8.53.3.14 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::Execute-Algorithm is called. Also the loading of the problem and algorithm configurations occurs in this method. That way, if it is called more than once and the algorithm or problem has changed, it will be accounted for.

Implements Optimizer.

8.53.3.15 bool accepts_multiple_points () const [virtual]

Overridden to return true since JEGA algorithms can accept multiple initial points.

Returns:

true, always.

Reimplemented from Iterator.

8.53.3.16 bool returns_multiple_points () const [virtual]

Overridden to return true since JEGA algorithms can return multiple final points.

Returns:

true, always.

Reimplemented from Iterator.
8.53 JEGAOptimizer Class Reference

8.53.3.17 void initial_points (const VariablesArray & pts) [virtual]

Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables.

Parameters:

The array of initial points for the JEGA algorithm created and run by this JEGAOptimizer.

Reimplemented from Iterator.

8.53.3.18 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.

8.53.4 Member Data Documentation

8.53.4.1 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.

The documentation for this class was generated from the following files:

- JEGAOptimizer.H
- JEGAOptimizer.C
8.54  JEGAOptimizer::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.

8.54.1  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.

8.54.2  Constructor & Destructor Documentation

8.54.2.1  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.
8.54.3 Member Function Documentation

8.54.3.1 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.

8.54.3.2 DesignOFSortSet PerformIterations (GeneticAlgorithm * theGA) [inline]

 Performs the required iterations on the supplied GA.

 This includes the calls to AlgorithmInitialize and AlgorithmFinalize and logs some information if appropriate.

 This is just here to expose the base class method to users.

 Parameters:

 theGA The GA on which to perform iterations. This parameter must be non-null.

 Returns:

 The final solutions reported by the supplied GA after all iterations and call to AlgorithmFinalize.

8.54.3.3 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:**

- *data* The algorithm and associated logger that are no longer needed and thus must be destroyed.

The documentation for this class was generated from the following file:

- [JEGAOptimizer.C](#)
8.55  JEGAOptimizer::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)
  
  Overriden to return a newly created Evaluator.

- EvaluatorCreator (Model &theModel)
  
  Constructs an EvaluatorCreator using the supplied model and optimizer.

Private Attributes

- Model & _theModel
  
  The user defined model to be passed to the constructor of the Evaluator.

8.55.1 Detailed Description

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

8.55.2 Constructor & Destructor Documentation

8.55.2.1 EvaluatorCreator (Model & theModel) [inline]

Constructs an EvaluatorCreator using the supplied model and optimizer.

Parameters:

  theModel  The Dakota::Model this creator will pass to the created evaluator.

  theOptimizer  The JEGAOptimizer this creator will pass to the created evaluator.
virtual GeneticAlgorithmEvaluator* CreateEvaluator (GeneticAlgorithm & \textit{alg}) [inline, virtual]

Overriden to return a newly created Evaluator.
The GA will assume ownership of the evaluator so we needn’t worry about keeping track of it for destruction. The additional parameters needed by the Evaluator are stored as members of this class at construction time.

Parameters:

\textit{alg}  The GA for which the evaluator is to be created.

Returns:

A pointer to a newly created Evaluator.

The documentation for this class was generated from the following file:

- JEGAOptimizer.C
8.56 LaguerreOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Laguerre polynomials.
Inheritance diagram for LaguerreOrthogPolynomial:

```
OrthogonalPolynomial

LaguerreOrthogPolynomial
```

Public Member Functions

- `LaguerreOrthogPolynomial ()`
  `default constructor`

- `~LaguerreOrthogPolynomial ()`
  `destructor`

Protected Member Functions

- `const Real & get_value (const Real &x, size_t n)`
  `retrieve the Laguerre polynomial value for a given parameter x`

- `const Real & get_gradient (const Real &x, size_t n)`
  `retrieve the Laguerre polynomial gradient for a given parameter x`

- `const Real & norm_squared (size_t n)`
  `return the inner product <L_n,L_n> = ||L_n||^2`

- `const RealVector & gauss_points (size_t n)`
  `polynomial order n`

- `const RealVector & gauss_weights (size_t n)`
  `polynomial order n`
8.56.1 Detailed Description

Derived orthogonal polynomial class for Laguerre polynomials.

The `LaguerreOrthogPolynomial` class evaluates a univariate Laguerre polynomial of a particular order. These polynomials are orthogonal with respect to the weight function \( \exp(-x) \) when integrated over the support range of \([0, +\infty)\). This corresponds to the probability density function for the standard exponential distribution. It enables (mixed) multidimensional orthogonal polynomial basis functions within `OrthogPolyApproximation`. Laguerre polynomials are a special case (\( \alpha = 0 \)) of the generalized Laguerre polynomials (implemented separately) which correspond to the standard gamma distribution.

The documentation for this class was generated from the following files:

- `LaguerreOrthogPolynomial.H`
- `LaguerreOrthogPolynomial.C`
8.57 LeastSq Class Reference

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 ()**
  *destructor*

- **void run ()**
  *run the iterator; portion of run_iterator()*

- **void print_results (ostream &s) const**

- **void derived_initialize_scaling (StringArray &fn_scale_types, RealVector &fn_scales)**
  *respectively*

- **void derived_post_run ()**

- **virtual void minimize_residuals ()=0**
  *for the least squares branch.*

- **void read_observed_data ()**
  *read user data file to load observed data points*
**Static Protected Member Functions**

- static void `primary_resp_recast` (const `Variables` &native_vars, const `Variables` &scaled_vars, const `Response` &native_response, `Response` &scaled_response)
  
  *from native (user) to iterator space*

**Protected Attributes**

- int `numLeastSqTerms`
  
  *number of least squares terms*

- String `obsDataFilename`
  
  *filename from which to read observed data*

- bool `obsDataFlag`
  
  *flag indicating whether user-supplied data is active*

- `RealVector obsData`
  
  *storage for user-supplied data for computing residuals*

**Static Protected Attributes**

- static `LeastSq` *`leastSqInstance`
  
  *pointer to `LeastSq` instance used in static member functions*

**8.57.1 Detailed Description**

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

The `LeastSq` class provides common data and functionality for `NLSSOLLeastSq` and `SNLLLeastSq`.

**8.57.2 Constructor & Destructor Documentation**

**8.57.2.1 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.
8.57.3 Member Function Documentation

8.57.3.1 void run () [inline, protected, virtual]

run the iterator; portion of run_iterator()

Iterator supports a construct/pre-run/run/post-run/destroy progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

8.57.3.2 void print_results (ostream & s) const [protected, virtual]

Redefines default iterator results printing to include nonlinear least squares results (residual terms and constraints).

Reimplemented from Iterator.

8.57.3.3 void derived_post_run () [protected, virtual]

Implements portions of post_run specific to LeastSq for scaling back to native variables and functions

Reimplemented from Iterator.

Reimplemented in SNLLLeastSq.

8.57.3.4 void primary_resp_recast (const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response) [static, protected]

from native (user) to iterator space

Least squares function map from user/native space to iterator/scaled space using a RecastModel. If no scaling also copies constraints.

8.57.3.5 void read_observed_data () [protected]

read user data file to load observed data points

read user’s observation data for computation of least squares residuals (currently reading on all processors – need to read once and broadcast)

The documentation for this class was generated from the following files:

- DakotaLeastSq.H
- DakotaLeastSq.C
8.58 LegendreOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Legendre polynomials.

Inheritance diagram for LegendreOrthogPolynomial:

```
OrthogonalPolynomial
    └── LegendreOrthogPolynomial
```

### Public Member Functions

- **LegendreOrthogPolynomial ()**
  
  *default constructor*

- **~LegendreOrthogPolynomial ()**
  
  *destructor*

### Protected Member Functions

- const Real & **get_value** (const Real &x, size_t n)
  
  *retrieve the Legendre polynomial value for a given parameter x*

- const Real & **get_gradient** (const Real &x, size_t n)
  
  *retrieve the Legendre polynomial gradient for a given parameter x*

- const Real & **norm_squared** (size_t n)
  
  *return the inner product $\langle P_n, P_n \rangle = ||P_n||^2$*

- const RealVector & **gauss_points** (size_t n)

  *polynomial order n*

- const RealVector & **gauss_weights** (size_t n)

  *polynomial order n*
8.58.1 Detailed Description

Derived orthogonal polynomial class for Legendre polynomials.

The LegendreOrthogPolynomial class evaluates a univariate Legendre polynomial of a particular order. These polynomials are orthogonal with respect to the weight function $1$ when integrated over the support range of $[-1,+1]$. This corresponds to the probability density function $f(x) = 1/(U-L) = 1/2$ for the uniform distribution for $[L,U]=[\text{-}1,1]$. It enables (mixed) multidimensional orthogonal polynomial basis functions within OrthogPolyApproximation. Legendre polynomials are a special case ($\alpha = \beta = 0$) of the more general Jacobi polynomials (implemented separately) which correspond to the beta distribution.

The documentation for this class was generated from the following files:

- LegendreOrthogPolynomial.H
- LegendreOrthogPolynomial.C
8.59  List Class Template Reference

Template class for the Dakota bookkeeping list.

Public Member Functions

- **List ()**
  *Default constructor.*

- **List (const List< T > &a)**
  *Copy constructor.*

- **~List ()**
  *Destructor.*

- **template<class InputIter> List (InputIter first, InputIter last)**
  *Range constructor (member template).*

- **List< T > & operator= (const List< T > &a)**
  *Assignment operator*

- **void write (ostream &s) const**
  *Writes a List to an output stream.*

- **void read (MPIUnpackBuffer &s)**
  *Reads a List from an MPIUnpackBuffer after an MPI receive.*

- **void write (MPIPackBuffer &s) const**
  *Writes a List to a MPIPackBuffer prior to an MPI send.*

- **size_t entries () const**
  *Returns the number of items that are currently in the list.*

- **T get ()**
  *Removes and returns the first item in the list.*

- **T removeAt (size_t index)**
  *Removes and returns the item at the specified index.*

- **bool remove (const T &a)**
  *Removes the specified item from the list.*
8.59 List Class Template Reference

- void `insert` (const T &a)
  
  *Adds the item a to the end of the list.*

- bool `contains` (const T &a) const
  
  *Returns TRUE if list contains object a, returns FALSE otherwise.*

- bool `find` (bool(*test_fn)(const T &, const void *), const void *test_fn_data, T &found_item) const
  
  *Function finds and sets k to this object*

- List<T>::iterator `find` (bool(*test_fn)(const T &, const void *), const void *test_fn_data)
  
  *Function finds*

- size_t `index` (bool(*test_fn)(const T &, const void *), const void *test_fn_data) const
  
  *Returns the index of object that the user defined test function finds.*

- size_t `index` (const T &a) const
  
  *Returns the index of the object.*

- size_t `count` (const T &a) const
  
  *Returns the number of items in the list equal to object.*

- T & `operator[]` (size_t i)
  
  *Returns the object at index i (can use as lvalue).*

- const T & `operator[]` (size_t i) const
  
  *Returns the object at index i, const (can’t use as lvalue).*

8.59.1 Detailed Description

`template<class T> class Dakota::List<T>`

Template class for the Dakota bookkeeping list.

The `List` is the common list class for Dakota. It inherits from either the RW list class or the STL list class. Extends the base list class to add Dakota specific methods Builds upon the previously existing DakotaValList class

8.59.2 Member Function Documentation

8.59.2.1 T get ()

Removes and returns the first item in the list.
Remove and return item from front of list. Returns the object pointed to by the list::begin() iterator. It also deletes the first node by calling the list::pop_front() method. Note: get() is not the same as list::front() since the latter would return the 1st item but would not delete it.

8.59.2.2  T removeAt (size_t index)

Removes and returns the item at the specified index.
Removes the item at the index specified. Uses the STL advance() function to step to the appropriate position in the list and then calls the list::erase() method.

8.59.2.3  bool remove (const T & a)

Removes the specified item from the list.
Removes the first instance matching object a from the list (and therefore differs from the STL list::remove() which removes all instances). Uses the STL find() algorithm to find the object and the list::erase() method to perform the remove.

8.59.2.4  void insert (const T & a)  [inline]

Adds the item a to the end of the list.
Insert item at end of list, calls list::push_back() method.

8.59.2.5  bool contains (const T & a)  const  [inline]

Returns TRUE if list contains object a, returns FALSE otherwise.
Uses the STL find() algorithm to locate the first instance of object a. Returns true if an instance is found.

8.59.2.6  bool find (bool(*)(const T &, const void * test_fn, const void * test_fn_data, T & found_item) const

function finds and sets k to this object
Find the first item in the list which satisfies the test function. Sets k if the object is found.

8.59.2.7  List<T>::iterator find (bool(*)(const T &, const void * test_fn, const void * test_fn_data) const

function finds
Find the first item in the list which satisfies the test function and return an iterator pointing to it.

8.59.2.8  size_t index (bool(*)(const T &, const void * test_fn, const void * test_fn_data) const

Returns the index of object that the user defined test function finds.
Returns the index of the first item in the list which satisfies the test function. Uses a single list traversal to both locate the object and return its index (generic algorithms would require two loop traversals).

8.59.2.9 size_t index (const T & a) const

Returns the index of the object.

 Returns the index of the first item in the list which matches the object a. Uses a single list traversal to both locate the object and return its index (generic algorithms would require two loop traversals).

8.59.2.10 size_t count (const T & a) const [inline]

Returns the number of items in the list equal to object.

Uses the STL count() algorithm to return the number of occurrences of the specified object.

8.59.2.11 ]

T & operator[ ] (size_t i)

Returns the object at index i (can use as lvalue).

Returns item at position i of the list by stepping through the list using forward or reverse STL iterators (depending on which end of the list is closer to the desired item). Once the object is found, it returns the value pointed to by the iterator.

This functionality is inefficient in 0-\textgreater l len loop-based list traversals and is being replaced by iterator-based list traversals in the main DAKOTA code. For isolated look-ups of a particular index, however, this approach is acceptable.

8.59.2.12 ]

const T & operator[ ] (size_t i) const

Returns the object at index i, const (can’t use as lvalue).

Returns const item at position i of the list by stepping through the list using forward or reverse STL iterators (depending on which end of the list is closer to the desired item). Once the object is found it returns the value pointed to by the iterator.

This functionality is inefficient in 0-\textgreater l len loop-based list traversals and is being replaced by iterator-based list traversals in the main DAKOTA code. For isolated look-ups of a particular index, however, this approach is acceptable.

The documentation for this class was generated from the following file:

- DakotaList.H
8.60 Matrix Class Template Reference

Template class for the Dakota numerical matrix.

Inheritance diagram for Matrix::

```
BaseVector< Dakota::BaseVector< T > >
```

```
Matrix
```

Public Member Functions

- **Matrix** (size_t num_rows=0, size_t num_cols=0)
  
  Constructor; takes number of rows, and number of columns as arguments.

- ~**Matrix** ()
  
  Destructor.

- **Matrix**< T > & operator= (const T &ival)
  
  Sets all elements in the matrix to ival.

- size_t num_rows () const
  
  Returns the number of rows for the matrix.

- size_t num_columns () const
  
  Returns the number of columns for the matrix.

- void reshape_2d (size_t num_rows, size_t num_cols)
  
  Resizes the matrix to num_rows by num_cols.

- void read (istream &s, size_t nr, size_t nc)
  
  Reads a portion of the Matrix from an input stream.

- void read (istream &s)
  
  Reads the complete Matrix from an input stream.

- void read_row_vector (istream &s, size_t i, size_t nc)
  
  Reads a portion of the ith Matrix row vector from an input stream.

- void read_row_vector (istream &s, size_t i)
  
  Reads the ith Matrix row vector from an input stream.
**8.60 Matrix Class Template Reference**

- **void write**(ostream &s, size_t nr, size_t nc, bool brackets, bool row_rtn, bool final_rtn) const
  
  Writes a portion of the Matrix to an output stream.

- **void write**(ostream &s, bool brackets, bool row_rtn, bool final_rtn) const
  
  Writes the complete Matrix to an output stream.

- **void write_row_vector**(ostream &s, size_t i, size_t nc, bool brackets, bool break_line, bool final_rtn) const
  
  Writes a portion of the ith Matrix row vector to an output stream.

- **void write_row_vector**(ostream &s, size_t i, bool brackets, bool break_line, bool final_rtn) const
  
  Writes the ith Matrix row vector to an output stream.

- **void read**(BiStream &s, size_t nr, size_t nc)
  
  Reads a portion of the Matrix from a binary input stream.

- **void read**(BiStream &s)
  
  Reads the complete Matrix from a binary input stream.

- **void read_row_vector**(BiStream &s, size_t i, size_t nc)
  
  Reads a portion of the ith Matrix row vector from a binary input stream.

- **void read_row_vector**(BiStream &s, size_t i)
  
  Reads the ith Matrix row vector from a binary input stream.

- **void write**(BoStream &s, size_t nr, size_t nc) const
  
  Writes a portion of the Matrix to a binary output stream.

- **void write**(BoStream &s) const
  
  Writes the complete Matrix to a binary output stream.

- **void write_row_vector**(BoStream &s, size_t i, size_t nc) const
  
  Writes a portion of the ith Matrix row vector to a binary output stream.

- **void write_row_vector**(BoStream &s, size_t i) const
  
  Writes the ith Matrix row vector to a binary output stream.

- **void read**(MPIUnpackBuffer &s)
  
  Reads a Matrix from an MPIUnpackBuffer after an MPI receive.

- **void read.annotated**(MPIUnpackBuffer &s)
  
  Reads an annotated Matrix from an MPIUnpackBuffer after an MPI receive.

- **void read_row_vector**(MPIUnpackBuffer &s, size_t i)
  
  Reads the ith Matrix row vector from an MPIUnpackBuffer after an MPI recv.
- void write (MPIPackBuffer &s) const
  
  Writes a Matrix to a MPIPackBuffer prior to an MPI send.

- void write_annotated (MPIPackBuffer &s) const
  
  Writes an annotated Matrix to a MPIPackBuffer prior to an MPI send.

- void write_row_vector (MPIPackBuffer &s, size_t i) const
  
  Writes the ith Matrix row vector to a MPIPackBuffer prior to an MPI send.

### 8.60.1 Detailed Description

**template<class T> class Dakota::Matrix<T>**

Template class for the Dakota numerical matrix.

A matrix class template to provide 2D arrays of objects. The matrix is zero-based, rows: 0 to (numRows-1) and cols: 0 to (numColumns-1). The class supports overloading of the subscript operator allowing it to emulate a normal built-in 2D array type. **Matrix** relies on the **BaseVector** template class to manage any differences between underlying DAKOTA_BASE_VECTOR implementations (RW, STL, etc.).

### 8.60.2 Member Function Documentation

#### 8.60.2.1 Matrix<T> & operator= (const T & val) [inline]

Sets all elements in the matrix to ival.

calls base class operator=(ival)

Reimplemented from **BaseVector**.

The documentation for this class was generated from the following file:

- DakotaMatrix.H
8.61 MergedConstraints Class Reference

the merged data view.
Inheritance diagram for MergedConstraints::

```
Constraints   VariablesUtil
    |        |
    |        MergedConstraints
    |
```

Public Member Functions

- **MergedConstraints ()**
  - default constructor

- **MergedConstraints (const ProblemDescDB &problem_db, const pair< short, short > &view)**
  - standard constructor

- **~MergedConstraints ()**
  - destructor

- **const RealVector & continuous_lower_bounds () const**
  - return the active continuous variable lower bounds

- **void continuous_lower_bounds (const RealVector &c_l_bnds)**
  - set the active continuous variable lower bounds

- **const RealVector & continuous_upper_bounds () const**
  - return the active continuous variable upper bounds

- **void continuous_upper_bounds (const RealVector &c_u_bnds)**
  - set the active continuous variable upper bounds

- **const RealVector & inactive_continuous_lower_bounds () const**
  - return the inactive continuous lower bounds

- **void inactive_continuous_lower_bounds (const RealVector &i_c_l_bnds)**
  - set the inactive continuous lower bounds

- **const RealVector & inactive_continuous_upper_bounds () const**
  - return the inactive continuous upper bounds

  

DAKOTA Version 4.1 Developers Manual generated on October 11, 2007
void inactive_continuous_upper_bounds (const RealVector &i_c_u_bnds)
   set the inactive continuous upper bounds

RealVector all_continuous_lower_bounds () const
   returns a single array with all continuous lower bounds

void all_continuous_lower_bounds (const RealVector &a_c_l_bnds)
   sets all continuous lower bounds using a single array

RealVector all_continuous_upper_bounds () const
   returns a single array with all continuous upper bounds

void all_continuous_upper_bounds (const RealVector &a_c_u_bnds)
   sets all continuous upper bounds using a single array

void write (ostream &s) const
   write a variable constraints object to an ostream

void read (istream &s)
   read a variable constraints object from an istream

Protected Member Functions

void copy_rep (const Constraints *con_rep)
   Used by copy() to copy the contents of a letter class.

void reshape_rep (const Sizet2DArray &vars_comps)
   Used by reshape(Sizet2DArray&) to rehape the contents of a letter class.

Private Attributes

- RealVector mergedDesignLowerBnds
  domains (integer values promoted to reals)

- RealVector mergedDesignUpperBnds
  domains (integer values promoted to reals)

- RealVector uncertainLowerBnds
  uncertain to merge)

- RealVector uncertainUpperBnds
  uncertain to merge)
8.61 MergedConstraints Class Reference

- `RealVector mergedStateLowerBnds
  domains (integer values promoted to reals)

- `RealVector mergedStateUpperBnds
  domains (integer values promoted to reals)

8.61.1 Detailed Description

the merged data view.

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The `MergedConstraints` derived class combines continuous and discrete domain types but separates design, uncertain, and state variable types. The result is merged design bounds arrays (`mergedDesignLowerBnds`, `mergedDesignUpperBnds`), uncertain distribution bounds arrays (`uncertainLowerBnds`, `uncertainUpperBnds`), and merged state bounds arrays (`mergedStateLowerBnds`, `mergedStateUpperBnds`). The branch and bound strategy uses this approach (see `Variables::get_variables(problem_db)` for variables type selection; variables type is passed to the `Constraints` constructor in `Model`).

8.61.2 Constructor & Destructor Documentation

8.61.2.1 `MergedConstraints` (const `ProblemDescDB` & `problem_db`, const pair< short, short > & `view`)

standard constructor

In this class, a merged 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: BranchBndStrategy. Extract fundamental lower and upper bounds and merge continuous and discrete domains to create `mergedDesignLowerBnds`, `mergedDesignUpperBnds`, `mergedStateLowerBnds`, and `mergedStateUpperBnds` using utilities from `VariablesUtil` (uncertain distribution bounds do not require any merging).

The documentation for this class was generated from the following files:

- MergedConstraints.H
- MergedConstraints.C
8.62 MergedVariables Class Reference

merged data view.

Inheritance diagram for MergedVariables:

```
    Variables
   / | \
  /  |  \
/    |   \
MergedVariables
```

Public Member Functions

- **MergedVariables ()**
  *default constructor*

- **MergedVariables (const ProblemDescDB &problem_db, const pair < short, short > &view)**
  *standard constructor*

- **~MergedVariables ()**
  *destructor*

- **size_t tv () const**
  *Returns total number of vars.*

- **const IntArray & merged_discrete_ids () const**
  *returns the list of discrete variables merged into a continuous array*

- **const RealVector & continuous_variables () const**
  *return the active continuous variables*

- **void continuous_variables (const RealVector &c_vars)**
  *set the active continuous variables*

- **const StringArray & continuous_variable_labels () const**
  *return the active continuous variable labels*

- **void continuous_variable_labels (const StringArray &c_v_labels)**
  *set the active continuous variable labels*

- **const RealVector & inactive_continuous_variables () const**
  *return the inactive continuous variables*
void inactive_continuous_variables (const RealVector &i_c_vars)
set the inactive continuous variables

const StringArray & inactive_continuous_variable_labels () const
return the inactive continuous variable labels

void inactive_continuous_variable_labels (const StringArray &i_c_v_labels)
set the inactive continuous variable labels

size_t acv () const
returns total number of continuous vars

RealVector all_continuous_variables () const
returns a single array with all continuous variables

void all_continuous_variables (const RealVector &a_c_vars)
sets all continuous variables using a single array

StringArray all_continuous_variable_labels () const
returns a single array with all continuous variable labels

void all_continuous_variable_labels (const StringArray &a_c_v_labels)
sets all continuous variable labels using a single array

StringArray all_variable_labels () const
returns a single array with all variable labels

void read (istream &s)
read a variables object from an istream

void write (ostream &s) const
write a variables object to an ostream

void write_aprepro (ostream &s) const
write a variables object to an ostream in aprepro format

void read_annotated (istream &s)
read a variables object in annotated format from an istream

void write_annotated (ostream &s) const
write a variables object in annotated format to an ostream

void write_tabular (ostream &s) const
write a variables object in tabular format to an ostream
void read (BiStream &s)
    read a variables object from the binary restart stream

void write (BoStream &s) const
    write a variables object to the binary restart stream

void read (MPIUnpackBuffer &s)
    read a variables object from a packed MPI buffer

void write (MPIPackBuffer &s) const
    write a variables object to a packed MPI buffer

Protected Member Functions

void copy_rep (const Variables *vars_rep)
    Used by copy() to copy the contents of a letter class.

void reshape_rep (const Sizet2DArray &vars_comps)
    Used by reshape() to reshape the contents of a letter class.

Private Member Functions

void build_types_ids ()
    construct VarTypes and VarIds arrays using variablesComponents

Private Attributes

RealVector mergedDesignVars
    domains (discrete values promoted to continuous)

RealVector uncertainVars
    the uncertain variables array (no discrete uncertain to merge)

RealVector mergedStateVars
    domains (discrete values promoted to continuous)

StringArray mergedDesignLabels
    a label array combining continuous and discrete design labels

StringArray uncertainLabels
    the uncertain variables label array (no discrete uncertain to combine)
8.62 MergedVariables Class Reference

- **StringArray mergedStateLabels**
  
a label array combining continuous and discrete state labels

- **IntArray mergedDiscreteIds**
  
  requirement is relaxed by merging them into a continuous array

**Friends**

- `bool operator==(const MergedVariables &vars1, const MergedVariables &vars2)`

  equality operator

**8.62.1 Detailed Description**

merged data view.

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The **MergedVariables** derived class combines continuous and discrete domain types but separates design, uncertain, and state variable types. The result is a single continuous array of design variables (mergedDesignVars), a single continuous array of uncertain variables (uncertainVars), and a single continuous array of state variables (mergedStateVars). The branch and bound strategy uses this approach (see Variables::get_variables(problem_db)).

**8.62.2 Constructor & Destructor Documentation**

**8.62.2.1 MergedVariables (const ProblemDescDB & problem_db, const pair< short, short > & view)**

standard constructor

In this class, a merged 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: BranchBndStrategy. Extract fundamental variable types and labels and merge continuous and discrete domains to create mergedDesignVars, mergedStateVars, mergedDesignLabels, and mergedStateLabels using utilities from VariablesUtil (uncertain variables and labels do not require any merging).

The documentation for this class was generated from the following files:

- MergedVariables.H
- MergedVariables.C
8.63 Minimizer Class Reference

Inheritance diagram for Minimizer:

```
  Iterator
     
      Minimizer

  LeastSq
     
      NL2SOLLeastSq

      NLSSOLLeastSq

      SNLLLeastSq

  Optimizer
     
      COLINOptimizer

      CONMINOptimizer

      DOTOptimizer

      EffGlobalOptimizer

      JEGAOptimizer

      NCSUOptimizer

      NLPQLPOptimizer

      NPSOLOptimizer

      SNLLOptimizer
```

Public Member Functions

- `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 () const`
return multiple final iterator solutions (variables)

- const ResponseArray & response_array_results () const
  return multiple final iterator solutions (response)

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

- ~Minimizer ()
  destructor

- void response_results_active_set (const ActiveSet &set)
  set the requested data for the final iterator response results

- virtual void derived_initialize_scaling (StringArray &fn_scale_types, RealVector &fn_scales)=0
  respectively

- void initialize_scaling ()
  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, IntVector &scale_types, RealVector &scale_mults, RealVector &scale_offsets)
  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
• RealVector modify_n2s (const RealVector &native_vars, const IntVector &scale_types, const RealVector &multipliers, const RealVector &offsets) const
  general RealVector mapping from native to scaled variables vectors:

• RealVector modify_s2n (const RealVector &scaled_vars, const IntVector &scale_types, const RealVector &multipliers, const RealVector &offsets) const
  general RealVector mapping from scaled to native variables:

• void response_modify_n2s (const Variables &scaled_vars, const Response &native_response, Response &scaled_response, int native_offset, int recast_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

Static Protected Member Functions

• static void variables_recast (const Variables &scaled_vars, Variables &native_vars)
  variables from scaled to native (user) space

• static void secondary_resp_recast (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &scaled_response)
  transform constraints (fns, grads, Hessians) from native (user) to

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
- `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

- `bool boundConstraintFlag`
  constraints. Used for method selection and error checking.

- `bool speculativeFlag`
  flag for speculative gradient evaluations

- `size_t numUserFnsLsqt`
  number of objective functions of least squares terms in the user’s model

- `size_t numIterFnsLsqt`
  number of objective functions of least squares terms in iterator’s view

- `bool scaleFlag`
  flag indicating scaling status

- `IntVector cvScaleTypes`
  scale flags for continuous vars.

- `RealVector cvScaleMultipliers`
  scales for continuous variables

- `RealVector cvScaleOffsets`
  offsets for continuous variables

- `IntVector 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)
- **IntVector linearIneqScaleTypes**  
  `scale flags for linear ineq`
- **RealVector linearIneqScaleMultipliers**  
  `scales for linear ineq constrs.`
- **RealVector linearIneqScaleOffsets**  
  `offsets for linear ineq constrs.`
- **IntVector linearEqScaleTypes**  
  `scale flags for linear eq.`
- **RealVector linearEqScaleMultipliers**  
  `scales for linear constraints`
- **RealVector linearEqScaleOffsets**  
  `offsets for linear constraints`
- **bool vendorNumericalGradFlag**  
  `convenience flag for gradType == numerical & methodSource == vendor`
- **Variables bestVariables**  
  `best variables found in minimization`
- **RealVector bestFunctions**  
  `best function values found in minimization; used in multiobjective/scaling cases`
- **Response bestResponses**  
  `best responses found in minimization`
- **VariablesArray bestVariablesArray**  
  `collection of all best solution variables.`
- **ResponseArray bestResponseArray**  
  `collection of all best solution responses.`

**Static Protected Attributes**

- **static Minimizer * minimizerInstance**  
  `pointer to Minimizer used in static member functions`
Friends

- class SOLBase
  
  access to iterator hierarchy data (to avoid attribute replication)

- class SNLLBase
  
  access to iterator hierarchy data (to avoid attribute replication)

8.63.1 Detailed Description

iterator hierarchy.

The Minimizer class provides common data and functionality for Optimizer and LeastSq.

8.63.2 Constructor & Destructor Documentation

8.63.2.1 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.

8.63.3 Member Function Documentation

8.63.3.1 void initialize_scaling () [protected]

checking

helper function used in constructors of derived classes to set up scaling types, multipliers and offsets when input scaling flag is enabled; includes call to the derived class’ derived_initialize_scaling()

8.63.3.2 void variables_recast (const Variables & scaled_vars, Variables & native_vars) [static, protected]

variables from scaled to native (user) space

Variables map from iterator/scaled space to user/native space using a RecastModel.
8.63.3.3 void secondary_resp_recast (const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response) [static, protected]

transform constraints (fns, grads, Hessians) from native (user) to
Constraint function map from user/native space to iterator/scaled/combined space using a RecastModel.

8.63.3.4 RealVector modify_n2s (const RealVector & native_vars, const IntVector & scale_types, const RealVector & multipliers, const RealVector & offsets) const [protected]

general RealVector mapping from native to scaled variables vectors:
general RealVector mapping from native to scaled variables; loosely, in greatest generality: scaled_var = log( (native_var - offset) / multiplier )

8.63.3.5 RealVector modify_s2n (const RealVector & scaled_vars, const IntVector & scale_types, const RealVector & multipliers, const RealVector & offsets) const [protected]

general RealVector mapping from scaled to native variables:
general RealVector mapping from scaled to native variables; loosely, in greatest generality: scaled_var = (LOG_BASE^scaled_var) * multiplier + offset

8.63.3.6 void response_modify_n2s (const Variables & native_vars, const Response & native_response, Response & recast_response, int native_offset, int recast_offset, int num_responses) const [protected]

map responses from native to scaled variable space
scaling response mapping: modifies response from a model (user/native) for use in iterators (scaled) – not including multi_objective_modify

8.63.3.7 RealMatrix lin_coeffs_modify_n2s (const RealMatrix & src_coeffs, const RealVector & cv_multipliers, const RealVector & lin_multipliers) const [protected]

general linear coefficients mapping from native to scaled space
compute scaled linear constraint matrix given design variable multipliers and linear scaling multipliers. Only scales components corresponding to continuous variables so for src_coeffs of size MxN, lin_multipliers.size() <= M, cv_multipliers.size() <= N

The documentation for this class was generated from the following files:

- DakotaMinimizer.H
- DakotaMinimizer.C
8.64 Model Class Reference

Base class for the model class hierarchy.

Inheritance diagram for Model::

```
Model
  NestedModel
  RecastModel
  SingleModel
  SurrogateModel
  DataFitSurrModel
  HierarchSurrModel
```

Public Member Functions

- **Model ()**
  
  *default constructor*

- **Model (ProblemDescDB &problem_db)**
  
  *standard constructor for envelope*

- **Model (const Model &model)**
  
  *copy constructor*

- virtual ~Model ()
  
  *destructor*

- **Model operator= (const Model &model)**
  
  *assignment operator*

- virtual Iterator & subordinate_iterator ()
  
  *return the sub-iterator in nested and surrogate models*

- virtual Model & 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** ()
  *or NestedModel::optionalInterface*

- virtual void **surrogate_bypass** (bool bypass_flag)
  *models contained within this model*

- 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 Response &response)
  *response at vars*

- virtual void **update_approximation** (const Variables &vars, const Response &response, bool rebuild_flag)
  *update an existing surrogate model with a new anchor*

- virtual void **update_approximation** (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  *update an existing surrogate model with new data points*

- virtual void **append_approximation** (const Variables &vars, const Response &response, bool rebuild_flag)
  *append a single point to an existing surrogate model’s data*

- virtual void **append_approximation** (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  *append multiple points to an existing surrogate model’s data*

- virtual **Array< Approximation >** & **approximations** ()
  *retrieve the set of Approximations within a DataFitSurrModel*

- virtual const RealVectorArray & **approximation_coefficients** ()
  *within a DataFitSurrModel*

- virtual void **approximation_coefficients** (const RealVectorArray &approx_coeffs)
  *within a DataFitSurrModel*

- virtual void **print_coefficients** (ostream &s, size_t index) const
  *within a DataFitSurrModel*

- virtual const RealVector & **approximation_variances** (const RealVector &c_vars)
  *Approximation within a DataFitSurrModel.*
• virtual const List< SurrogateDataPoint > & approximation_data (size_t index)
  instance within a DataFitSurfModel

• virtual void compute_correction (const Response &truth_response, const Response &approx_response,
  const RealVector &c_vars)
  compute correction factors for use in SurrogateModels

• virtual void auto_correction (bool correction_flag)
  manages automatic application of correction factors in SurrogateModels

• virtual bool auto_correction ()
  model’s responses

• virtual void apply_correction (Response &approx_response, const RealVector &c_vars, bool quiet_flag=false)
  apply correction factors to approx_response (for use in SurrogateModels)

• virtual void component_parallel_mode (short mode)
  (SUB_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 ()
  a termination message is received from stop_servers().

• virtual void stop_servers ()
  particular model when iteration on the model is complete.

• virtual bool derived_master_overload () const
  of trying to run a multiprocessor job on the master.

• virtual const String & interface_id () const
  return the interface identifier

• virtual int evaluation_id () const
  Return the current function evaluation id for the Model.

• virtual void set_evaluation_reference ()
  Set the reference points for the evaluation counters within the Model.
• virtual void print_evaluation_summary (ostream &s, bool minimal_header=false, bool relative_count=true) const
  
  Print an evaluation summary for the Model.

• 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 ()
  
  Response at currentVariables (default ActiveSet).

• void asynch_compute_response (const ActiveSet &set)
  
  Response at currentVariables (specified ActiveSet).

• const ResponseArray & synchronize ()
  
  complete set of results from a group of asynchronous evaluations.

• const IntResponseMap & synchronize_nowait ()
  
  available results from a group of asynchronous evaluations.

• void init_communicators (const int &max_iterator_concurrency, bool recurse_flag=true) const
  
  configuration in modelPCIterMap

• void init_serial ()
  
  modify some default settings to behave properly in serial.

• void set_communicators (const int &max_iterator_concurrency, bool recurse_flag=true) from modelPCIterMap

• void free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  
  deallocate communicator partitions for a model

• 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
  
  return total number of vars
- `size_t cv () const``
  return number of active continuous variables

- `size_t dv () const``
  return number of active discrete variables

- `size_t icv () const``
  return number of inactive continuous variables

- `size_t idv () const``
  return number of inactive discrete variables

- `size_t acv () const``
  return total number of continuous variables

- `size_t adv () const``
  return total number of discrete variables

- `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

- `const IntVector & discrete_variables () const``
  return the active discrete variables from currentVariables

- `void discrete_variables (const IntVector &d_vars)``
  set the active discrete variables in currentVariables

- `const RealVector & inactive_continuous_variables () const``
  return the inactive continuous variables in currentVariables

- `void inactive_continuous_variables (const RealVector &i_c_vars)``
  set the inactive continuous variables in currentVariables

- `const IntVector & inactive_discrete_variables () const``
  return the inactive discrete variables in currentVariables

- `void inactive_discrete_variables (const IntVector &i_d_vars)``
  set the inactive discrete variables in currentVariables

- `RealVector all_continuous_variables () const`
return all continuous variables in currentVariables

- void all_continuous_variables (const RealVector &a_c_vars)
  set all continuous variables in currentVariables

- IntVector all_discrete_variables () const
  return all discrete variables in currentVariables

- void all_discrete_variables (const IntVector &a_d_vars)
  set all discrete variables in currentVariables

- const RealVector & normal_means () const
  return the normal uncertain variable means

- void normal_means (const RealVector &n_means)
  set the normal uncertain variable means

- const RealVector & normal_std_deviations () const
  return the normal uncertain variable standard deviations

- void normal_std_deviations (const RealVector &n_std_devs)
  set the normal uncertain variable standard deviations

- const RealVector & normal_lower_bounds () const
  return the normal uncertain variable lower bounds

- void normal_lower_bounds (const RealVector &n_lower_bnds)
  set the normal uncertain variable lower bounds

- const RealVector & normal_upper_bounds () const
  return the normal uncertain variable upper bounds

- void normal_upper_bounds (const RealVector &n_upper_bnds)
  set the normal uncertain variable upper bounds

- const RealVector & lognormal_means () const
  return the lognormal uncertain variable means

- void lognormal_means (const RealVector &ln_means)
  set the lognormal uncertain variable means

- const RealVector & lognormal_std_deviations () const
  return the lognormal uncertain variable standard deviations

- void lognormal_std_deviations (const RealVector &ln_std_devs)
  set the lognormal uncertain variable standard deviations
• const RealVector & lognormal_error_factors () const
  return the lognormal uncertain variable error factors

• void lognormal_error_factors (const RealVector &ln_err_facts)
  set the lognormal uncertain variable error factors

• const RealVector & lognormal_lower_bounds () const
  return the lognormal uncertain variable lower bounds

• void lognormal_lower_bounds (const RealVector &ln_lower_bnds)
  set the lognormal uncertain variable lower bounds

• const RealVector & lognormal_upper_bounds () const
  return the lognormal uncertain variable upper bounds

• void lognormal_upper_bounds (const RealVector &ln_upper_bnds)
  set the lognormal uncertain variable upper bounds

• const RealVector & uniform_lower_bounds () const
  return the uniform uncertain variable lower bounds

• void uniform_lower_bounds (const RealVector &u_lower_bnds)
  set the uniform uncertain variable lower bounds

• const RealVector & uniform_upper_bounds () const
  return the uniform uncertain variable upper bounds

• void uniform_upper_bounds (const RealVector &u_upper_bnds)
  set the uniform uncertain variable upper bounds

• const RealVector & loguniform_lower_bounds () const
  return the loguniform uncertain variable lower bounds

• void loguniform_lower_bounds (const RealVector &lu_lower_bnds)
  set the loguniform uncertain variable lower bounds

• const RealVector & loguniform_upper_bounds () const
  return the loguniform uncertain variable upper bounds

• void loguniform_upper_bounds (const RealVector &lu_upper_bnds)
  set the loguniform uncertain variable upper bounds

• const RealVector & triangular_modes () const
  return the triangular uncertain variable modes
void triangular_modes (const RealVector &t_modes)
    set the triangular uncertain variable modes

const RealVector & triangular_lower_bounds () const
    return the triangular uncertain variable lower bounds

void triangular_lower_bounds (const RealVector &t_lower_bnds)
    set the triangular uncertain variable lower bounds

const RealVector & triangular_upper_bounds () const
    return the triangular uncertain variable upper bounds

void triangular_upper_bounds (const RealVector &t_upper_bnds)
    set the triangular uncertain variable upper bounds

const RealVector & exponential_betas () const
    return the exponential uncertain variable beta parameters

void exponential_betas (const RealVector &e_betas)
    set the exponential uncertain variable beta parameters

const RealVector & beta_alphas () const
    return the beta uncertain variable alphas

void beta_alphas (const RealVector &b_alphas)
    set the beta uncertain variable alphas

const RealVector & beta_betas () const
    return the beta uncertain variable betas

void beta_betas (const RealVector &b_betas)
    set the beta uncertain variable betas

const RealVector & beta_lower_bounds () const
    return the beta uncertain variable lower bounds

void beta_lower_bounds (const RealVector &b_lower_bnds)
    set the beta uncertain variable lower bounds

const RealVector & beta_upper_bounds () const
    return the beta uncertain variable upper bounds

void beta_upper_bounds (const RealVector &b_upper_bnds)
    set the beta uncertain variable upper bounds

const RealVector & gamma_alphas () const
return the gamma uncertain variable alpha parameters

- void gamma_alphas (const RealVector &ga_alphas)
  set the gamma uncertain variable alpha parameters

- const RealVector & gamma_betas () const
  return the gamma uncertain variable beta parameters

- void gamma_betas (const RealVector &ga_betas)
  set the gamma uncertain variable beta parameters

- const RealVector & gumbel_alphas () const
  return the gumbel uncertain variable alphas

- void gumbel_alphas (const RealVector &gu_alphas)
  set the gumbel uncertain variable alphas

- const RealVector & gumbel_betas () const
  return the gumbel uncertain variable betas

- void gumbel_betas (const RealVector &gu_betas)
  set the gumbel uncertain variable betas

- const RealVector & frechet_alphas () const
  return the frechet uncertain variable alpha parameters

- void frechet_alphas (const RealVector &f_alphas)
  set the frechet uncertain variable alpha parameters

- const RealVector & frechet_betas () const
  return the frechet uncertain variable beta parameters

- void frechet_betas (const RealVector &f_betas)
  set the frechet uncertain variable beta parameters

- const RealVector & weibull_alphas () const
  return the weibull uncertain variable alpha parameters

- void weibull_alphas (const RealVector &w_alphas)
  set the weibull uncertain variable alpha parameters

- const RealVector & weibull_betas () const
  return the weibull uncertain variable beta parameters

- void weibull_betas (const RealVector &w_betas)
  set the weibull uncertain variable beta parameters
• const RealVectorArray & histogram_bin_pairs () const
  return the histogram uncertain bin pairs

• void histogram_bin_pairs (const RealVectorArray &h_bin_pairs)
  set the histogram uncertain bin pairs

• const RealVectorArray & histogram_point_pairs () const
  return the histogram uncertain point pairs

• void histogram_point_pairs (const RealVectorArray &h_pt_pairs)
  set the histogram uncertain point pairs

• const RealVectorArray & interval_probabilities () const
  return the interval basic probability values

• void interval_probabilities (const RealVectorArray &int_probs)
  set the interval basic probability values

• const RealVectorArray & interval_bounds () const
  return the interval bounds

• void interval_bounds (const RealVectorArray &int_bounds)
  set the interval bounds

• const RealMatrix & uncertain_correlations () const
  return the uncertain variable correlations

• void uncertain_correlations (const RealMatrix &uncertain_corr)
  set the uncertain variable correlations

• const StringArray & continuous_variable_types () const
  return the active continuous variable types from currentVariables

• const StringArray & discrete_variable_types () const
  return the active discrete variable types from currentVariables

• const StringArray & continuous_variable_labels () const
  return the active continuous variable labels from currentVariables

• void continuous_variable_labels (const StringArray &c_v_labels)
  set the active continuous variable labels in currentVariables

• const StringArray & discrete_variable_labels () const
  return the active discrete variable labels from currentVariables
• void discrete_variable_labels (const StringArray &d_v_labels)
  set the active discrete variable labels in currentVariables

• const StringArray & inactive_continuous_variable_labels () const
  return the inactive continuous variable labels in currentVariables

• void inactive_continuous_variable_labels (const StringArray &i_c_v_labels)
  set the inactive continuous variable labels in currentVariables

• const StringArray & inactive_discrete_variable_labels () const
  return the inactive discrete variable labels in currentVariables

• void inactive_discrete_variable_labels (const StringArray &i_d_v_labels)
  set the inactive discrete variable labels in currentVariables

• StringArray all_continuous_variable_labels () const
  return all continuous variable labels in currentVariables

• void all_continuous_variable_labels (const StringArray &a_c_v_labels)
  set all continuous variable labels in currentVariables

• StringArray all_discrete_variable_labels () const
  return all discrete variable labels in currentVariables

• void all_discrete_variable_labels (const StringArray &a_d_v_labels)
  set all discrete variable 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

• 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

• const IntVector & discrete_lower_bounds () const
return the active discrete lower bounds from userDefinedConstraints

- void discrete_lower_bounds (const IntVector &d_l_bnds)
  set the active discrete lower bounds in userDefinedConstraints

- const IntVector & discrete_upper_bounds () const
  return the active discrete upper bounds from userDefinedConstraints

- void discrete_upper_bounds (const IntVector &d_u_bnds)
  set the active discrete upper bounds 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_lower_bounds () const
  return the inactive discrete lower bounds in userDefinedConstraints

- void inactive_discrete_lower_bounds (const IntVector &i_d_l_bnds)
  set the inactive discrete lower bounds in userDefinedConstraints

- const IntVector & inactive_discrete_upper_bounds () const
  return the inactive discrete upper bounds in userDefinedConstraints

- void inactive_discrete_upper_bounds (const IntVector &i_d_u_bnds)
  set the inactive discrete upper bounds in userDefinedConstraints

- RealVector all_continuous_lower_bounds () const
  return all continuous lower bounds in userDefinedConstraints

- void all_continuous_lower_bounds (const RealVector &a_c_l_bnds)
  set all continuous lower bounds in userDefinedConstraints

- RealVector all_continuous_upper_bounds () const
  return all continuous upper bounds in userDefinedConstraints

- void all_continuous_upper_bounds (const RealVector &a_c_u_bnds)
  set all continuous upper bounds in userDefinedConstraints
- `IntVector all_discrete_lower_bounds() const`
  
  return all discrete lower bounds in userDefinedConstraints

- `void all_discrete_lower_bounds(const IntVector &a_d_l_bnds)`
  
  set all discrete lower bounds in userDefinedConstraints

- `IntVector all_discrete_upper_bounds() const`
  
  return all discrete upper bounds in userDefinedConstraints

- `void all_discrete_upper_bounds(const IntVector &a_d_u_bnds)`
  
  set all discrete upper bounds in userDefinedConstraints

- `size_t num_linear_ineq_constraints() const`
  
  return the number of linear inequality constraints

- `size_t num_linear_eq_constraints() const`
  
  return the number of linear equality constraints

- `const RealMatrix & linear_ineq_constraint_coeffs() const`
  
  return the linear inequality constraint coefficients

- `void linear_ineq_constraint_coeffs(const RealMatrix &lin_ineq_coeffs)`
  
  set the linear inequality constraint coefficients

- `const RealVector & linear_ineq_constraint_lower_bounds() const`
  
  return the linear inequality constraint lower bounds

- `void linear_ineq_constraint_lower_bounds(const RealVector &lin_ineq_l_bnds)`
  
  set the linear inequality constraint lower bounds

- `const RealVector & linear_ineq_constraint_upper_bounds() const`
  
  return the linear inequality constraint upper bounds

- `void linear_ineq_constraint_upper_bounds(const RealVector &lin_ineq_u_bnds)`
  
  set the linear inequality constraint upper bounds

- `const RealMatrix & linear_eq_constraint_coeffs() const`
  
  return the linear equality constraint coefficients

- `void linear_eq_constraint_coeffs(const RealMatrix &lin_eq_coeffs)`
  
  set the linear equality constraint coefficients

- `const RealVector & linear_eq_constraint_targets() const`
  
  return the linear equality constraint targets
void linear_eq_constraint_targets (const RealVector &lin_eq_targets)
    set the linear equality constraint targets

size_t num_nonlinear_ineq_constraints () const
    return the number of nonlinear inequality constraints

size_t num_nonlinear_eq_constraints () const
    return the number of nonlinear equality constraints

const RealVector & nonlinear_ineq_constraint_lower_bounds () const
    return the nonlinear inequality constraint lower bounds

void nonlinear_ineq_constraint_lower_bounds (const RealVector &nln_ineq_l_bnds)
    set the nonlinear inequality constraint lower bounds

const RealVector & nonlinear_ineq_constraint_upper_bounds () const
    return the nonlinear inequality constraint upper bounds

void nonlinear_ineq_constraint_upper_bounds (const RealVector &nln_ineq_u_bnds)
    set the nonlinear inequality constraint upper bounds

const RealVector & nonlinear_eq_constraint_targets () const
    return the nonlinear equality constraint targets

void nonlinear_eq_constraint_targets (const RealVector &nln_eq_targets)
    set the nonlinear equality constraint targets

const IntArray & merged_discrete_ids () const
    merged into a continuous array in currentVariables

const Variables & current_variables () const
    return the current variables (currentVariables)

const Constraints & user_defined_constraints () const
    return the user-defined constraints (userDefinedConstraints)

const Response & current_response () const
    return the current response (currentResponse)

const ProblemDescDB & problem_description_db () const
    return the problem description database (probDescDB)

const String & model_type () const
    return the model type (modelType)

const String & model_id () const
return the model identifier (idModel)

- 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)

- const RealVector & fd_gradient_step_size () const
  return the finite difference gradient step size (fdGradSS)

- const IntList & gradient_id_analytic () const
  return the mixed gradient analytic IDs (gradIdAnalytic)

- const IntList & 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 IntList & hessian_id_analytic () const
  return the mixed Hessian analytic IDs (hessIdAnalytic)

- const IntList & hessian_id_numerical () const
  return the mixed Hessian analytic IDs (hessIdNumerical)

- const IntList & hessian_id_quasi () const
  return the mixed Hessian analytic IDs (hessIdQuasi)

- void supports_estimated_derivatives (bool sed_flag)
  set whether this model should perform or pass on derivative estimation
• const 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 ag)
  set the asynchronous evaluation flag (asynchEvalFlag)

• 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 ag)
  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
  that are not mapped to the top Model level

Protected Member Functions

• Model (BaseConstructor, ProblemDescDB &problem_db)
  derived class constructors - Coplien, p. 139)

• Model (NoDBBaseConstructor, const pair< short, short > &view, const ActiveSet &set)
  constructed on the fly

• Model (RecastBaseConstructor, ParallelLibrary &parallel_lib)
  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 ResponseArray & 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 (const 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 (const int &max_iterator_concurrency, bool recurse_flag=true)  
portion of set_communicators() specific to derived model classes

virtual void derived_free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)  
portion of free_communicators() specific to derived model classes

Protected Attributes

- Variables currentVariables  
  function evaluations

- size_t numDerivVars  
  corrections where only the active continuous variables are supported)

- Response currentResponse  
  function evaluations

- size_t numFns  
  the number of functions in currentResponse

- Constraints userDefinedConstraints  
  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*

- RealVector fdGradSS
  *relative step sizes for numerical gradients*

- IntList gradIdAnalytic
  *analytic id’s for mixed gradients*

- IntList 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*

- IntList hessIdAnalytic
  *analytic id’s for mixed Hessians*

- IntList hessIdNumerical
  *numerical id’s for mixed Hessians*

- IntList hessIdQuasi
  *quasi id’s for mixed Hessians*

- bool supportsEstimDerivs
  *whether model should perform or forward derivative estimation*

- IntArray messageLengths
  *and PRPair*

- const ProblemDescDB & probDescDB
  *class member reference to the problem description database*
- **ParallelLibrary & parallelLib**
  class member reference to the parallel library

- **ParConfigIter modelPCIter**
  the ParallelConfiguration node used by this model instance

- **short componentParallelMode**
  (SUB_MODEL/HF_MODEL/TRUTH_MODEL)

- **bool asynchEvalFlag**
  flags asynch evaluations (local or distributed)

- **RealVector normalMeans**
  normal uncertain variable means

- **RealVector normalStdDevs**
  normal uncertain variable standard deviations

- **RealVector normalLowerBnds**
  normal uncertain variable lower bounds

- **RealVector normalUpperBnds**
  normal uncertain variable upper bounds

- **RealVector lognormalMeans**
  lognormal uncertain variable means

- **RealVector lognormalStdDevs**
  lognormal uncertain variable standard deviations

- **RealVector lognormalErrFacts**
  lognormal uncertain variable error factors

- **RealVector lognormalLowerBnds**
  lognormal uncertain variable lower bounds

- **RealVector lognormalUpperBnds**
  lognormal uncertain variable upper bounds

- **RealVector uniformLowerBnds**
  uniform uncertain variable lower bounds

- **RealVector uniformUpperBnds**
  uniform uncertain variable upper bounds
- `RealVector loguniformLowerBnds`
  loguniform uncertain variable lower bounds

- `RealVector loguniformUpperBnds`
  loguniform uncertain variable upper bounds

- `RealVector triangularModes`
  triangular uncertain variable modes

- `RealVector triangularLowerBnds`
  triangular uncertain variable lower bounds

- `RealVector triangularUpperBnds`
  triangular uncertain variable upper bounds

- `RealVector exponentialBetas`
  exponential uncertain variable betas

- `RealVector betaAlphas`
  beta uncertain variable alphas

- `RealVector betaBetas`
  beta uncertain variable betas

- `RealVector betaLowerBnds`
  beta uncertain variable lower bounds

- `RealVector betaUpperBnds`
  beta uncertain variable upper bounds

- `RealVector gammaAlphas`
  gamma uncertain variable alphas

- `RealVector gammaBetas`
  gamma uncertain variable betas

- `RealVector gumbelAlphas`
  gumbel uncertain variable alphas

- `RealVector gumbelBetas`
  gumbel uncertain variable betas

- `RealVector frechetAlphas`
  frechet uncertain variable alphas

- `RealVector frechetBetas`
frechet uncertain variable betas

- RealVector weibullAlphas
  weibull uncertain variable alphas

- RealVector weibullBetas
  weibull uncertain variable betas

- RealVectorArray histogramBinPairs
  histogram uncertain (x,y) bin pairs (continuous linear histogram)

- RealVectorArray histogramPointPairs
  histogram uncertain (x,y) point pairs (discrete histogram)

- RealVectorArray intervalBasicProbs
  basic probability values for interval uncertain variables

- RealVectorArray intervalBounds
  interval lower/upper bounds for interval uncertain variables

- RealMatrix uncertainCorrelations
  and correlation coefficients for reliability

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_,
  method_source) in the numerical gradient specification.

- void synchronize_derivatives (const Variables &vars, const ResponseArray &fd_responses, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set)
  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 RealMatrixArray &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

- 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 idModel
  model identifier string from the input file

- bool estDerivsFlag
  asynch_compute_response()

- int evaluationCapacity
  capacity for concurrent evaluations supported by the Model

- std::map< int, ParConfigIter > modelPCIterMap
  level as the lookup key

- bool modelAutoGraphicsFlag
  graphics posting at the strategy level)

- bool silentFlag
  flag for really quiet (silent) model output

- bool quietFlag
  flag for quiet model output

- ModelList modelList
  used to collect sub-models for subordinate_models()

- VariablesList varsList
  synchronize().

- List< ShortArray > asvList
  asynch_compute_response() to synchronize()

- List< ActiveSet > setList
  asynch_compute_response() to synchronize()

- BoolList initialMapList
  synchronize_derivatives()

- BoolList dbCaptureList
8.64 Model Class Reference

 synchronizes derivatives()

- **ResponseList** dbResponseList
  synchronizes derivatives()

- **RealList** deltaList
  transfers deltas from estimate_derivatives() to synchronize_derivatives()

- **IntList** numMapsList
  into numerical gradients.

- **RealMatrix** 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

- **RealMatrixArray** quasiHessians
  quasi-Newton Hessian approximations

- **SizetArray** numQuasiUpdates
  number of quasi-Newton Hessian updates applied

- **ResponseArray** responseArray
  similar array in Interface contains the raw responses.

- **IntResponseMap** graphicsRespMap
  prior to sequential input into the graphics

- **Model** * modelRep
  pointer to the letter (initialized only for the envelope)

- **int** referenceCount
  number of objects sharing modelRep

8.64.1 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.
8.64.2 Constructor & Destructor Documentation

8.64.2.1 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.

8.64.2.2 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.

8.64.2.3 Model (const Model & model)

copy constructor

Copy constructor manages sharing of modelRep and incrementing of referenceCount.

8.64.2.4 ~Model () [virtual]

destructor

Destructor decrements referenceCount and only deletes modelRep when referenceCount reaches zero.

8.64.2.5 Model (BaseConstructor, ProblemDescDB & problem_db) [protected]

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).

8.64.2.6 Model (RecastBaseConstructor, ParallelLibrary & parallel_lib) [protected]

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.
8.64.3 Member Function Documentation

8.64.3.1 Model operator= (const Model & model)

assignment operator

8.64.3.2 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, and NestedModel.

8.64.3.3 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, and HierarchSurrModel.

8.64.3.4 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, and HierarchSurrModel.

8.64.3.5 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 recursion will stop as soon as it encounters a Model that was instantiated normally, which is appropriate since ProblemDescDB-constructed Models use top-down information flow and do not require bottom-up updating.
Reimplemented in DataFitSurrModel, and RecastModel.
8.64.3.6 **Interface & interface()**  [virtual]

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`.

8.64.3.7 **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`.

8.64.3.8 **int local_eval_concurrency()**  [virtual]

return derived model asynchronous evaluation concurrency

SingleModels and HierarchSurrModels redefine this virtual function.

Reimplemented in `RecastModel`, and `SingleModel`.

8.64.3.9 **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`.

8.64.3.10 **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).

8.64.3.11 **void init_communicators(const int & max_iterator_concurrency, bool recurse_flag = true)**

configuration in modelPCIterMap
The `init_communicators()` and `derived_init_communicators()` functions are structured to avoid performing the messageLengths estimation more than once. `init_communicators()` (not virtual) performs the estimation and then forwards the results to `derived_init_communicators` (virtual) which uses the data in different contexts.

**8.64.3.12 void init_serial()**

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.

**8.64.3.13 void estimate_message_lengths()**

estimate messageLengths for a model

This functionality has been pulled out of `init_communicators()` and defined separately so that it may be used in those cases when messageLengths is needed but `model.init_communicators()` is not called, e.g., for the master processor in the self-scheduling of a concurrent iterator strategy.

**8.64.3.14 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).

**8.64.3.15 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.
8.64.3.16 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.

8.64.3.17 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]

method_source) in the numerical gradient specification.
Estimate derivatives by computing finite difference gradients, finite difference Hessians, and/or quasi-Newton Hessians. The total number of finite difference evaluations is returned for use by synchronize() to track response arrays, and it could be used to improve management of max_function_evaluations within the iterators.

8.64.3.18 void synchronize_derivatives (const Variables & vars, const ResponseArray & fd_responses, const Response & new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set) [private]

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().

8.64.3.19 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, const RealMatrix & initial_map_response, const RealMatrixArray & 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.

8.64.3.20 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.
8.64.3.21 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.

The documentation for this class was generated from the following files:

- DakotaModel.H
- DakotaModel.C
8.65 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.

8.65.1 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.H
- MPIPackBuffer.C
8.66 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.

8.66.1 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.H
- MPIPackBuffer.C
8.67 MultilevelOptStrategy Class Reference

multiple models of varying fidelity.

Inheritance diagram for MultilevelOptStrategy::

```
Strategy

MultilevelOptStrategy
```

Public Member Functions

- **MultilevelOptStrategy** (ProblemDescDB &problem_db)
  
  constructor

- **~MultilevelOptStrategy** ()
  
  destructor

- void **run_strategy** ()
  
  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 Member Functions

- void **run_coupled** ()
  
  run a tightly coupled hybrid

- void **run_uncoupled** ()
  
  run an uncoupled hybrid

- void **run_uncoupled_adaptive** ()
  
  run an uncoupled adaptive hybrid
Private Attributes

- **String multiLevelType**
  
  coupled, uncoupled, or uncoupled_adaptive

- **StringArray methodList**
  
  the list of method identifiers

- **int numIterators**
  
  number of methods in methodList

- **size_t numSolutionsTransferred**
  
  to the next iterator

- **Real localSearchProb**
  
  phases of the global optimization for coupled hybrids

- **Real progressMetric**
  
  an uncoupled adaptive hybrid

- **Real progressThreshold**
  
  uncoupled adaptive hybrid switches to the next method

- **IteratorArray selectedIterators**
  
  the set of iterators, one for each entry in methodList

- **ModelArray userDefinedModels**
  
  the set of models, one for each iterator

### 8.67.1 Detailed Description

multiple models of varying fidelity.

This strategy has three approaches to hybrid optimization: (1) the uncoupled 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; (2) the uncoupled adaptive hybrid is similar to the uncoupled hybrid, except that the stopping rules for the optimizers are controlled adaptively by the strategy instead of internally by each optimizer; and (3) the coupled hybrid uses multiple methods in close coordination, generally using a local search optimizer repeatedly within a global optimizer (the local search optimizer refines candidate optima which are fed back to the global optimizer). The uncoupled strategies only pass information forward, whereas the coupled strategy allows both feed forward and feedback. Note that while the strategy is targeted at optimizers, 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.

### 8.67.2 Member Function Documentation
8.67.2.1  void run_coupled ()  [private]

run a tightly coupled hybrid
In the coupled case, use is made of external hybridization capabilities, such as those available in the global/local hybrids from SGOPT. This function is responsible only for publishing the local optimizer selection to the global optimizer and then invoking the global optimizer; the logic of method switching is handled entirely within the global optimizer. Status: incomplete.

8.67.2.2  void run_uncoupled ()  [private]

run an uncoupled hybrid
In the uncoupled nonadaptive case, there is no interference with the iterators. Each runs until its own convergence criteria is satisfied. Status: fully operational.

8.67.2.3  void run_uncoupled_adaptive ()  [private]

run an uncoupled adaptive hybrid
In the uncoupled 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.

The documentation for this class was generated from the following files:

- MultilevelOptStrategy.H
- MultilevelOptStrategy.C
8.68 NCSUOptimizer Class Reference

Wrapper class for the NCSU DIRECT optimization library.

Inheritance diagram for NCSUOptimizer::

```
                      Iterator
                         |
                         v
                     Minimizer
                         |
                         v
                     Optimizer
                         |
                         v
NCSUOptimizer
```

Public Member Functions

- **NCSUOptimizer (Model &model)**
  - *standard constructor*

- **NCSUOptimizer (Model &model, const int &max_iterations, const int &max_fn_evals)**
  - *alternate constructor for instantiations "on the fly"*

- **NCSUOptimizer (const RealVector &var_lower_bnds, const RealVector &var_upper_bnds, const int &max_iterations, const int &max_fn_eval, void(*user_obj_eval)(int &, double *, double &, int &, int *, int &, double *, int &, char *, int &))**
  - *alternate constructor for instantiations "on the fly"*

- **~NCSUOptimizer ()**
  - *destructor*

- **void find_optimum ()**
  - *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 &n, double *x, double &f, int &flag, int *idata, int &isize, double *ddata, int &dsize, char *cdata, int &csize)
  objective function (passed by function pointer to NCSUDirect).

Private Attributes

- String setUpType
  GaussProcApproximation currently uses the user_functions mode.

- int maxIterations
  holds maximum number of iterations allowed

- int maxFunctionEvals
  holds maximum number of function evaluations allowed

- Real minBoxSize
  holds the minimum boxsize

- Real volBoxSize
  hold the minimum volume boxsize

- Real solutionAccuracy
  holds the solution tolerance accuracy

- 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 &, double *, double &, int &, int *, int &, double *, int &, char *, int &)
  "user_functions" mode.

Static Private Attributes

- static NCSUOptimizer * ncsudirectInstance
  functions in order to avoid the need for static data
8.68 NCSUOptimizer Class Reference

8.68.1 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:

8.68.2 Constructor & Destructor Documentation

8.68.2.1 NCSUOptimizer (Model & model, const int & max_iterations, const int & max_fn_evals)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

8.68.2.2 NCSUOptimizer (const RealVector & var_lower_bnds, const RealVector & var_upper_bnds, const int & max_iterations, const int & max_fn_evals, void(*)(int &, double *, double &, int &, int *, int &, double *, int &, char *, int &) user_obj_eval)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function pointer.

The documentation for this class was generated from the following files:

- NCSUOptimizer.H
- NCSUOptimizer.C
8.69 NestedModel Class Reference

execution within every evaluation of the model.

Inheritance diagram for NestedModel:

```
Model

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NestedModel</td>
</tr>
<tr>
<td>----------------</td>
</tr>
</tbody>
</table>
```

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

- void **derived_subordinate_models** (ModelList &ml, bool recurse_flag)
  return subModel

- **Interface & interface** ()
  return optionalInterface

- void **surrogate_bypass** (bool bypass_flag)
  to the subModel for any lower-level surrogates.

- void **component_parallel_mode** (short mode)
optionalInterface and subModel

- bool derived_master_overload() const
evaluation (forwarded to optionalInterface)

- void derived_init_communicators(const 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(const int &max_iterator_concurrency, bool recurse_flag=true)
  set active parallel configuration within subModel

- void derived_free_communicators(const int &max_iterator_concurrency, bool recurse_flag=true)
  (forwarded to optionalInterface and subModel)

- void serve()
  stop_servers().

- void stop_servers()
  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()
  (request forwarded to optionalInterface and subModel)

- void print_evaluation_summary(ostream &s, bool minimal_header=false, bool relative_count=true) const
  (request forwarded to optionalInterface and subModel)

Private Member Functions

- void asv_mapping(const ShortArray &mapped_asv, ShortArray &interface_asv, ShortArray &sub_iterator_asv)
  total model evaluation requirements (mapped_asv)

- void response_mapping(const Response &interface_response, const Response &sub_iterator_response, Response &mapped_response)
  mappings to create the total response for the model
void update_sub_model ()
   
   update subModel with current variable values/bounds/labels

Private Attributes

- int nestedModelEvals
  
  derived_asynch_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
  
  sub-iteration results

- size_t numSubIterMappedEqCon
  
  sub-iteration results

- Interface optionalInterface
  
  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
  
  functions) resulting from optional interface evaluations

- size_t numOptInterfIneqCon
  
  interface evaluations

- size_t numOptInterfEqCon
  
  interface evaluations

- SizetArray primaryCVarMapIndices
  
  replace the subModel variable values.
8.69 NestedModel Class Reference

- **SizetArray primaryDVarMapIndices**
  
  insertions replace the subModel variable values.

- **SizetArray secondaryVarMapIndices**
  
  for uncertain variables) of the active continuous subModel variables.

- **size_t subModelCVOffset**
  
  mappings when the subModel is in an All variables view.

- **size_t subModelDVOffset**
  
  mappings when the subModel is in an All variables view.

- **RealMatrix primaryRespCoeffs**
  
  generic response terms.

- **RealMatrix secondaryRespCoeffs**
  
  contributions to the top-level inequality and equality constraints.

### 8.69.1 Detailed Description

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.

### 8.69.2 Member Function Documentation

#### 8.69.2.1 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.

#### 8.69.2.2 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.

8.69.2.3 bool derived_master_overload () const [inline, protected, virtual]

evaluation (forwarded to optionalInterface)

Derived master overload for subModel is handled separately in subModel.compute_response() within sub-Iterator.run().

Reimplemented from Model.

8.69.2.4 void derived_init_communicators (const 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.

8.69.2.5 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.

8.69.2.6 void response_mapping (const Response & opt_interface_response, const Response & sub_iterator_response, Response & mapped_response) [private]

mappings to create the total response for the model

In the OUU case,

optionalInterface fns = {f}, {g} (deterministic primary functions, constraints)
subIterator fns = {S} (UQ response statistics)

Problem formulation for mapped functions:

\[
\begin{align*}
\text{minimize} & \quad f + W(S) \\
\text{subject to} & \quad g_l \leq g \leq g_u \\
& \quad a_l \leq A(S) \leq a_u \\
& \quad g = g_t \\
& \quad A(S) = a_t
\end{align*}
\]
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: \([g_l] <= [g] + [A][S] <= [g_u]\) with \([A]\) usage the same as for \([W]\) above.

In the UOO case, things are simpler, just compute statistics of each optimization response function: \([W] = [I], \{f\}/\{g\}/[A]\) are empty.

### 8.69.3 Member Data Documentation

#### 8.69.3.1 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.

The documentation for this class was generated from the following files:

- NestedModel.H
- NestedModel.C
8.70 Nl2Misc Struct Reference

Auxiliary information passed to calcr and calcj via ur.

Public Attributes

- Real * J [2]
  *cache the two most recent Jacobian values in speculative-evaluation mode*

- int nf [2]
  *function-evaluation counts corresponding to cached Jacobian values (used to tell which J value to use)*

- int specgrad
  *whether to cache J values (0 == no, 1 == yes)*

8.70.1 Detailed Description

Auxiliary information passed to calcr and calcj via ur.

The documentation for this struct was generated from the following file:

- NL2SOLLeastSq.C
8.71 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 ()**
  *destructor*

- **void minimize_residuals ()**
  *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 = statprt (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 `dltfde`
  finite-diff step size for function differences for H (fd_hessian_step_size)

- int `mxcal`
  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 `xctol`
  x-convergence tolerance (x_conv_tol)

- Real `sctol`
  singular convergence tolerance (singular_conv_tol)

- Real `lmmaxs`
  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 ==> $\sigma^2 H^{-1} J^T J H^{-1} J$ 2 or -2 ==> $\sigma^2 H^{-1}$ 3 or -3 ==> $\sigma^2 (J^T J)^{-1}$ 1 or 2 ==> use gradient diffs to estimate H -1 or -2 ==> use function diffs to estimate H default = 0 (no covariance)
8.71 NL2SOLLeastSq Class Reference

- int rdreq
  * whether to compute the regression diagnostic vector (regression_diagnostics)

- Real fprec
  * expected response function precision (function_precision)

- Real lmax0
  * initial trust-region radius (initial_trust_radius)

**Static Private Attributes**

- static NL2SOLLeastSq * nl2solInstance
  * evaluator functions

### 8.71.1 Detailed Description

Wrapper class for the NL2SOL nonlinear least squares library.

The NL2SOLLeastSq class provides a wrapper for NL2SOL (TOMS Algorithm 573), in the updated form of Port Library routines dn[f][g][b] 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.

The documentation for this class was generated from the following files:

- NL2SOLLeastSq.H
- NL2SOLLeastSq.C
8.72 NLPQLPOptimizer Class Reference

Wrapper class for the NLPQLP optimization library, Version 2.0.

Inheritance diagram for NLPQLPOptimizer:

```
         Iterator
           ↓
        Minimizer
           ↓
          Optimizer
           ↓
       NLPQLPOptimizer
```

**Public Member Functions**

- `NLPQLPOptimizer (Model &model)`
  *
  constructor

- `~NLPQLPOptimizer ()`
  *
  destructor

- `void find_optimum ()`
  *
  Redefines the run virtual function for the optimizer branch.

**Protected Member Functions**

- `virtual void derived_pre_run ()`
  *
  performs run-time set up

**Private Member Functions**

- `void allocate_workspace ()`
  *
  Allocates workspace for the optimizer.

- `void deallocate_workspace ()`
  *
  Releases workspace memory.
• void allocate_constraints ()
  Allocates constraint mappings.

Private Attributes

• int L
  the serial version by setting L=1.

• int numEqConstraints
  numEqConstraints : Number of equality constraints.

• int MMAX
  MMAX must be at least one and greater or equal to M.

• int N
  N : Number of optimization variables.

• int NMAX
  than N.

• int MNN2
  MNN2 : Must be equal to M+N+N+2.

• double * X
  function values should be computed simultaneously.

• double * F
  values to be computed from L iterates stored in X.

• double * G
  function values to be computed from L iterates stored in X.

• double * DF
  of F to compute DF.

• double * DG
  has to be equal to MMAX.

• double * U
  inequality constraints should be nonnegative.

• double * C
  to NMAX.
- double * D
  array D.

- double ACC
  than the accuracy by which gradients are computed.

- double ACCQP
  by NLPQLP and subsequently multiplied by 1.0D+4.

- double STPMIN
  by STPMIN**(1/L-1). If STPMIN<0, then STPMIN=ACC is used.

- int MAXFUN
  than 50.

- int MAXIT
  gradients (e.g. 100).

- int MAX_NM
  MAX_NM=0, monotone line search is performed.

- double TOL_NM
  non-negative (e.g. 0.1).

- int IPRINT
  values are displayed during the line search.

- int MODE
  function in C and D in form of an LDL decomposition.

- int IOUT
  write-statements start with 'WRITE(IOUT,... '.

- int IFAIL
  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
8.72 NLPQLP Optimizer Class Reference

\[ \text{LKWA} : \text{LKWA should be at least } N+10. \]

- \text{int } \text{ACTIVE}  \\
  \text{ACTIVE}(J)=.TRUE., J=1,...,M.

- \text{int } \text{LACTIVE}  \\
  \text{least } 2+M+10.

- \text{int } \text{LQL}  \\
  \text{contains only an upper triangular factor.}

- \text{int } \text{numNlpqlConstr}  \\
  \text{total number of constraints seen by NLPQL}

- \text{SizetList } \text{nonlinIneqConMappingIndices}  \\
  \text{constraints used in computing the corresponding NLPQL constraints.}

- \text{RealList } \text{nonlinIneqConMappingMultipliers}  \\
  \text{constraints to the corresponding NLPQL constraints.}

- \text{RealList } \text{nonlinIneqConMappingOffsets}  \\
  \text{constraints to the corresponding NLPQL constraints.}

- \text{SizetList } \text{linIneqConMappingIndices}  \\
  \text{constraints used in computing the corresponding NLPQL constraints.}

- \text{RealList } \text{linIneqConMappingMultipliers}  \\
  \text{constraints to the corresponding NLPQL constraints.}

- \text{RealList } \text{linIneqConMappingOffsets}  \\
  \text{constraints to the corresponding NLPQL constraints.}

8.72.1 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

\[ \text{minimize } F(X) \text{ subject to } G(J,X) = 0 , J=1,...,\text{ME} \text{ } G(J,X) \geq 0 , J=\text{ME}+1,...,\text{M} \text{ } X_L \leq X \leq X_U \]

and is an extension of the code NLPQLD. NLPQLP is specifically tuned to run under distributed systems. A new input parameter \text{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 use 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.H
- NLPQLPOptimizer.C
8.73 NLSSOLLeastSq Class Reference

Wrapper class for the NLSSOL nonlinear least squares library.

Inheritance diagram for NLSSOLLeastSq:

```
  Iterator
   |
   Minimizer
   |
   LeastSq   SOLBase
   |
   NLSSOLLeastSq
```

Public Member Functions

- **NLSSOLLeastSq (Model &model)**
  _standard constructor_

- **~NLSSOLLeastSq ()**
  _destructor_

- **void minimize_residuals ()**
  _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)**
  _least squares terms (passed by function pointer to NLSSOL)._

Static Private Attributes

- **static NLSSOLLastSq * nlssolInstance**
  _functions in order to avoid the need for static data_
8.73.1  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.

The documentation for this class was generated from the following files:

- NLSSOLLeastSq.H
- NLSSOLLeastSq.C
8.74 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.

8.74.1 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:

- global_def.h
8.75 NonD Class Reference

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

Inheritance diagram for NonD:

```
NonD
  |   |
  |   |    NonDAnalyzer
  |   |    NonDAnalyzer
  |   |    NonDIterator
  |   |    NonDIter::
  |   |    NonDEvidence
  |   |    NonDIntegration
  |   |    NonDPolynomialChaos
  |   |    NonDReliability
  |   |    NonDSampling
  |   |    NonDCubature
  |   |    NonDQuadrature
  |   |    NonDGlobalReliability
  |   |    NonDLocalReliability
  |   |    NonDAdaptImpSampling
  |   |    NonDIncremLHSSampling
  |   |    NonDLHSSampling
```

Public Member Functions

- void `initialize_random_variables()`
  - `iteratedModel (corrCholeskyFactorZ and correlationFlagX set separately)`

- void `initialize_random_variables` (const ShortArray &x_types, const ShortArray &u_types, const Epetra_SerialDenseVector &x_means, const Epetra_SerialDenseVector &x_std_devs, const Epetra_SerialDenseVector &x_l_bnds, const Epetra_SerialDenseVector &x_u_bnds, const Array<Epetra_SerialDenseVector> &x_addtl, const Epetra_SerialDenseMatrix &z_chol_fact, bool x_corr_ag, bool ext_u_space)
  - `correlationFlagX, and extendedUSpace 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_target, bool cdf_ag)
  - `combination with alternate ctors`

- void `moments` (const RealVector &means, const RealVector &std_devs)
  - `set meanStats and stdDevStats`

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 run()`
  invoke `quantify_uncertainty`

- `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
  distributions into response statistics

- `virtual void initialize_final_statistics()`
  initializes `finalStatistics` for storing `NonD` final results

- `void initialize_random_variable_types()`
  initializes `ranVarTypesX` and `ranVarTypesU`

- `void initialize_random_variable_parameters()`
  `ranVarUpperBndsX`, and `ranVarAddtlParamsX`

- `void reshape_correlation_matrix()`
  reshape `corrMatrix` for an all_variables specification

- `void trans_U_to_X(const Epetra_SerialDenseVector &u_vars, Epetra_SerialDenseVector &x_vars)`
  variables to x-space of correlated random variables

- `void trans_U_to_X(const RealVector &u_vars, RealVector &x_vars)`
  Overloaded form using `RealVectors`.

- `void trans_U_to_Z(const Epetra_SerialDenseVector &u_vars, Epetra_SerialDenseVector &z_vars)`
  variables to z-space of correlated standard normal variables

- `void trans_Z_to_X(const Epetra_SerialDenseVector &z_vars, Epetra_SerialDenseVector &x_vars)`
  variables to x-space of correlated random variables

- `void trans_X_to_U(const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseVector &u_vars)`
  to u-space of uncorrelated standard normal variables

- `void trans_X_to_U(const RealVector &x_vars, RealVector &u_vars)`
  Overloaded form using `RealVectors`. 
• void `trans_X_to_Z` (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseVector &z_vars)
  to z-space of correlated standard normal variables

• void `trans_Z_to_U` (Epetra_SerialDenseVector &z_vars, Epetra_SerialDenseVector &u_vars)
  variables to u-space of uncorrelated standard normal variables

• void `trans_correlations` ()
  and decomposes it into its Cholesky factor (corrCholeskyFactorZ).

• void `trans_grad_X_to_U` (const Epetra_SerialDenseVector &fn_grad_x, Epetra_SerialDenseVector &fn_grad_u, const Epetra_SerialDenseVector &x_vars)
  Transformation routine for gradient vector from x-space to u-space.

• void `trans_grad_U_to_X` (const Epetra_SerialDenseVector &fn_grad_u, Epetra_SerialDenseVector &fn_grad_x, const Epetra_SerialDenseVector &x_vars)
  Transformation routine for gradient vector from u-space to x-space.

• void `trans_hess_X_to_U` (const Epetra_SerialSymDenseMatrix &fn_hess_x, Epetra_SerialSymDenseMatrix &fn_hess_u, const Epetra_SerialDenseVector &x_vars, const Epetra_SerialDenseVector &fn_grad_x)
  Transformation routine for Hessian matrix from x-space to u-space.

• void `jacobian_dX_dU` (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseMatrix &jacobian_xu)
  Jacobian of x(u) mapping obtained from dX/dZ dZ/dU.

• void `jacobian_dX_dZ` (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseMatrix &jacobian_xz)
  Jacobian of x(z) mapping obtained from differentiation of `trans_Z_to_X()`.

• void `jacobian_dU_dX` (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseMatrix &jacobian_ux)
  Jacobian of u(x) mapping obtained from dU/dZ dZ/dX.

• void `jacobian_dZ_dX` (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseMatrix &jacobian_zx)
  Jacobian of z(x) mapping obtained from differentiation of `trans_X_to_Z()`.

• void `hessian_d2X_dU2` (const Epetra_SerialDenseVector &x_vars, Array &hessian_xu)
  Hessian of x(u) mapping obtained from dZ/dU dU dZ dZ/dU.

• void `hessian_d2X_dZ2` (const Epetra_SerialDenseVector &x_vars, Array &hessian_xz)
  Hessian of x(z) mapping obtained from differentiation of `jacobian_dX_dZ()`.

• Real `phi` (const Real &beta)
Standard normal density function.

- **Real** \texttt{Phi} (const Real &beta)
  
  Standard normal cumulative distribution function.

- **Real** \texttt{Phi\_inverse} (const Real &p)
  
  Inverse of standard normal cumulative distribution function.

- **Real** \texttt{erf\_inverse} (const Real &p)
  
  Inverse of error function used in \texttt{Phi\_inverse}().

- **Real** \texttt{cdf\_beta\_Pinv} (const Real &normcdf, const Real &alpha, const Real &beta)
  
  Inverse of standard beta CDF (not supported by GSL).

**Static Protected Member Functions**

- static void \texttt{vars\_u\_to\_x\_mapping} (const Variables &u\_vars, Variables &x\_vars)
  
  for \texttt{NonD} Iterators to x-space variables for Models.

- static void \texttt{resp\_x\_to\_u\_mapping} (const Variables &x\_vars, const Variables &u\_vars, const Response &x\_-response, Response &u\_response)
  
  static function used to define the approximate subproblem constraints.

**Protected Attributes**

- bool \texttt{extendedUSpace}
  
  std uniforms, std exponentials, std betas, and std gammas.

- ShortArray \texttt{ranVarTypesX}
  
  vector of indices indicating the type of each x-space uncertain variable

- ShortArray \texttt{ranVarTypesU}
  
  which each x-space variable is transformed

- Epetra_SerialDenseVector \texttt{ranVarMeansX}
  
  vector of means for all x-space uncertain variables

- Epetra_SerialDenseVector \texttt{ranVarStdDevsX}
  
  vector of standard deviations for all x-space uncertain variables

- Epetra_SerialDenseVector \texttt{ranVarLowerBndsX}
  
  vector of distribution lower bounds for selected x-space uncertain vars
- **Epetra_SerialDenseVector** `ranVarUpperBndsX`
  vector of distribution upper bounds for selected x-space uncertain vars

- **Array** `<Epetra_SerialDenseVector>` `ranVarAddtlParamsX`
  for selected x-space uncertain variables

- **bool** `correlationFlagX`
  uncertain variables

- **Epetra_SerialDenseMatrix** `corrCholeskyFactorZ`
  in `trans_correlations()` for use in z-space)

- **Epetra_SerialSymDenseMatrix** `corrMatrix`
  Epetra copy of `Model::uncertainCorrelations`.

- **const Real** `Pi`
  the value for Pi used in several numerical routines

- **size_t** `numDesignVars`
  within design variable bounds for All view modes)

- **size_t** `numStateVars`
  within state variable bounds for All view modes)

- **size_t** `numNormalVars`
  number of normal uncertain variables

- **size_t** `numLognormalVars`
  number of lognormal uncertain variables

- **size_t** `numUniformVars`
  number of uniform uncertain variables

- **size_t** `numLoguniformVars`
  number of loguniform uncertain variables

- **size_t** `numTriangularVars`
  number of triangular uncertain variables

- **size_t** `numExponentialVars`
  number of exponential uncertain variables

- **size_t** `numBetaVars`
  number of beta uncertain variables

- **size_t** `numGammaVars`
nonD Class Reference

number of gamma uncertain variables

- size_t numGumbelVars
  number of gumbel uncertain variables

- size_t numFrechetVars
  number of frechet uncertain variables

- size_t numWeibullVars
  number of weibull uncertain variables

- size_t numHistogramVars
  number of histogram uncertain variables

- size_t numIntervalVars
  number of interval uncertain variables

- size_t numUncertainVars
  total number of uncertain variables

- size_t numResponseFunctions
  number of response functions

- RealVector meanStats
  means of response functions (calculated in compute_statistics())

- RealVector stdDevStats
  std deviations of response functions (calculated in compute_statistics())

- RealVectorArray requestedRespLevels
  requested response levels for all response functions

- RealVectorArray computedProbLevels
  from requestedRespLevels

- RealVectorArray computedRelLevels
  from requestedRespLevels

- RealVectorArray computedGenRelLevels
  resulting from requestedRespLevels

- short respLevelTarget
  or z->beta+ (GEN_RELIABILITIES)

- 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**
  requestedProbLevels, requestedRelLevels, or requestedGenRelLevels

- **size_t totalLevelRequests**
  requestedProbLevels, and requestedRelLevels

- **bool cdfFlag**
  cumulative/CDF (true) or complementary/CCDF (false)

- **Response finalStatistics**
  response means, standard deviations, and probabilities of failure

**Static Protected Attributes**

- static **NonD * nondInstance**
  functions in order to avoid the need for static data

**Private Member Functions**

- void **distribute_levels (RealVectorArray &levels)**
  response functions if a short-hand specification is employed.

**8.75.1 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.

**8.75.2 Member Function Documentation**
8.75.2.1 void initialize_random_variables ()

iteratedModel (corrCholeskyFactorZ and correlationFlagX set separately)
Build ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations.

8.75.2.2 void initialize_final_statistics () [protected, virtual]

initializes finalStatistics for storing NonD final results
Default definition of virtual function (used by sampling, reliability, and polynomial chaos) defines the set of statistical results to include means, standard deviations, and level mappings.
Reimplemented in NonDEvidence.

8.75.2.3 void initialize_random_variable_types () [protected]

initializes ranVarTypesX and ranVarTypesU
Build ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations.

8.75.2.4 void initialize_random_variable_parameters () [protected]

ranVarUpperBndsX, and ranVarAddtlParamsX
Build ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations.

8.75.2.5 void trans_U_to_X (const Epetra_SerialDenseVector & u_vars, Epetra_SerialDenseVector & x_vars) [protected]

variables to x-space of correlated random variables
This procedure performs the transformation from u to x space. u_vars is the vector of random variables in uncorrelated standard normal space (u-space). x_vars is the vector of random variables in the original user-defined x-space.

8.75.2.6 void trans_U_to_Z (const Epetra_SerialDenseVector & u_vars, Epetra_SerialDenseVector & z_vars) [protected]

variables to z-space of correlated standard normal variables
This procedure computes the transformation from u to z space. u_vars is the vector of random variables in uncorrelated standard normal space (u-space). z_vars is the vector of random variables in normal space with proper correlations (z-space).
8.75.2.7 void trans_Z_to_X (const Epetra_SerialDenseVector & z_vars, Epetra_SerialDenseVector & x_vars) [protected]

variables to x-space of correlated random variables

This procedure computes the transformation from z to x space. z_vars is the vector of random variables in normal space with proper correlations (z-space). x_vars is the vector of random variables in the original user-defined x-space.

8.75.2.8 void trans_X_to_U (const Epetra_SerialDenseVector & x_vars, Epetra_SerialDenseVector & u_vars) [protected]

variables to u-space of uncorrelated standard normal variables

This procedure performs the transformation from x to u space u_vars is the vector of random variables in uncorrelated standard normal space (u-space). x_vars is the vector of random variables in the original user-defined x-space.

8.75.2.9 void trans_X_to_Z (const Epetra_SerialDenseVector & x_vars, Epetra_SerialDenseVector & z_vars) [protected]

variables to z-space of correlated standard normal variables

This procedure performs the transformation from x to z space: z_vars is the vector of random variables in normal space with proper correlations (z-space). x_vars is the vector of random variables in the original user-defined x-space.

8.75.2.10 void trans_Z_to_U (Epetra_SerialDenseVector & z_vars, Epetra_SerialDenseVector & u_vars)

[protected]

variables to u-space of uncorrelated standard normal variables

This procedure computes the transformation from z to u space. u_vars is the vector of random variables in uncorrelated standard normal space (u-space). z_vars is the vector of random variables in normal space with proper correlations (z-space).

8.75.2.11 void trans_correlations () [protected]

and decomposes it into its Cholesky factor (corrCholeskyFactorZ).

This procedure modifies the correlation matrix input by the user for use in the Nataf distribution model (Der Kiureghian and Liu, ASCE JEM 112:1, 1986). It uses empirical expressionss derived from least-squares polynomial fits to numerical integration data.

- corrMatrix: the correlation coefficient matrix of the random variables provided by the user

- mod_corr_matrix: modified correlation matrix
• corrCholeskyFactorZ: Cholesky factor of the modified correlation matrix for use in Z_to_U and U_to_Z transformations.

Note: The modification is exact for normal-normal, lognormal-lognormal, and normal-lognormal transformations. All other cases are approximations with some error as noted below.

8.75.2.12 void trans_grad_X_to_U (const Epetra_SerialDenseVector & fn_grad_x, Epetra_SerialDenseVector & fn_grad_u, const Epetra_SerialDenseVector & x_vars) [protected]

Transformation routine for gradient vector from x-space to u-space.

This procedure transforms a gradient vector from the original user-defined x-space (where evaluations are performed) to uncorrelated standard normal space (u-space). x_vars is the vector of random variables in x-space.

8.75.2.13 void trans_grad_U_to_X (const Epetra_SerialDenseVector & fn_grad_u, Epetra_SerialDenseVector & fn_grad_x, const Epetra_SerialDenseVector & x_vars) [protected]

Transformation routine for gradient vector from u-space to x-space.

This procedure transforms a gradient vector from uncorrelated standard space (u-space) to the original user-defined x-space. x_vars is the vector of random variables in u-space.

8.75.2.14 void trans_hess_X_to_U (const Epetra_SerialSymDenseMatrix & fn_hess_x, Epetra_SerialSymDenseMatrix & fn_hess_u, const Epetra_SerialDenseVector & x_vars, const Epetra_SerialDenseVector & fn_grad_x) [protected]

Transformation routine for Hessian matrix from x-space to u-space.

This procedure transforms a Hessian matrix from the original user-defined x-space (where evaluations are performed) to uncorrelated standard normal space (u-space). x_vars is the vector of the random variables in x-space.

8.75.2.15 void jacobian_dX_dU (const Epetra_SerialDenseVector & x_vars, Epetra_SerialDenseMatrix & jacobian_xu) [protected]

Jacobian of x(u) mapping obtained from dX/dZ dZ/dU.

This procedure computes the Jacobian of the transformation x(u). x_vars is the vector of random variables in the original user-defined x-space.

8.75.2.16 void jacobian_dX_dZ (const Epetra_SerialDenseVector & x_vars, Epetra_SerialDenseMatrix & jacobian_xz) [protected]

Jacobian of x(z) mapping obtained from differentiation of trans_Z_to_X().
This procedure computes the Jacobian of the transformation \( x(z) \). \( x\_vars \) is the vector of random variables in the original user-defined x-space.

8.75.2.17  
\[
\text{void jacobian\_dU\_dX (const Epetra\_SerialDenseVector & x\_vars, Epetra\_SerialDenseMatrix & jacobian\_ux)} \quad [\text{protected}]
\]

Jacobian of \( u(x) \) mapping obtained from \( dU/dZ \) \( dZ/dX \).

This procedure computes the Jacobian of the transformation \( u(x) \). \( x\_vars \) is the vector of random variables in the original user-defined x-space.

8.75.2.18  
\[
\text{void jacobian\_dZ\_dX (const Epetra\_SerialDenseVector & x\_vars, Epetra\_SerialDenseMatrix & jacobian\_zx)} \quad [\text{protected}]
\]

Jacobian of \( z(x) \) mapping obtained from differentiation of \( \text{trans\_X\_to\_Z()} \).

This procedure computes the Jacobian of the transformation \( z(x) \). \( x\_vars \) is the vector of random variables in the original user-defined x-space.

8.75.2.19  
\[
\text{void hessian\_d2X\_dU2 (const Epetra\_SerialDenseVector & x\_vars, Array<< Epetra\_SerialSymDenseMatrix > & hessian\_xu)} \quad [\text{protected}]
\]

Hessian of \( x(u) \) mapping obtained from \( dZ/dU^T d^2X/dZ^2 dZ/dU \).

This procedure computes the Hessian of the transformation \( x(u) \). \( hessian\_xu \) is a 3D tensor modeled as an array of matrices, where the \( i\_th \) matrix is \( d^2X_i/dU^2 \). \( x\_vars \) is the vector of random variables in the original user-defined x-space.

8.75.2.20  
\[
\text{void hessian\_d2X\_dZ2 (const Epetra\_SerialDenseVector & x\_vars, Array<< Epetra\_SerialSymDenseMatrix > & hessian\_xz)} \quad [\text{protected}]
\]

Hessian of \( x(z) \) mapping obtained from differentiation of \( \text{jacobian\_dX\_dZ()} \).

This procedure computes the Hessian of the transformation \( x(z) \). \( hessian\_xz \) is a 3D tensor modeled as an array of matrices, where the \( i\_th \) matrix is \( d^2X_i/dZ^2 \). \( x\_vars \) is the vector of random variables in the original user-defined x-space.

8.75.2.21  
\[
\text{Real Phi (const Real & beta) \quad [inline, protected]}
\]

Standard normal cumulative distribution function.
returns a probability < 0.5 for negative beta and a probability > 0.5 for positive beta.

8.75.2.22  
\[
\text{Real Phi\_inverse (const Real & p) \quad [inline, protected]}
\]

Inverse of standard normal cumulative distribution function.
returns a negative beta for probability < 0.5 and a positive beta for probability > 0.5.
8.75.2.23 Real cdf_beta_Pinv (const Real & normcdf, const Real & alpha, const Real & beta) [protected]

Inverse of standard beta CDF (not supported by GSL).
Solve is performed in scaled space (for the standard beta distribution).
The documentation for this class was generated from the following files:

- DakotaNonD.H
- DakotaNonD.C
8.76 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, int samples, int seed, short sampling_type, const bool cdf_flag, const bool x_space_data, const bool x_space_model, const bool bounded_model)`
  constructor

- `~NonDAdaptImpSampling ()`
  destructor

- `void quantify_uncertainty ()`
  failure.

- `void initialize (const RealVectorArray &initial_points, int resp_fn, const Real &initial_prob, const Real &failure_threshold)`
  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)`
  initial probability to refine, and flags to control transformations

- `const Real & get_probability ()`
  returns the probability calculated by the importance sampling
Private Member Functions

- void converge_cov ()
  until coefficient of variation converges

- void converge_probability ()
  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)
  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**
  initial points

- **bool transPoints**
  before evaluation

- **bool useModelBounds**
  flag to control if the sampler should respect the model bounds

- **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.

### 8.76.1 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.

### 8.76.2 Member Function Documentation

#### 8.76.2.1 void initialize (const RealVectorArray & initial_points, int resp_fn, const Real & initial_prob, const Real & failure_threshold)

initial probability to refine, and flags to control transformations

Initializes data using a set of starting points.

#### 8.76.2.2 void initialize (const RealVector & initial_point, int resp_fn, const Real & initial_prob, const Real & failure_threshold)

initial probability to refine, and flags to control transformations
Initializes data using only one starting point.

The documentation for this class was generated from the following files:

- NonDAdaptImpSampling.H
- NonDAdaptImpSampling.C
8.77  NonDCubature Class Reference

integrals over uncorrelated uniforms.

Inheritance diagram for NonDCubature::

```
  Iterator
    Analyzer
      NonD
        NonDIntegration
          NonDCubature
```

Public Member Functions

- NonDCubature (Model &model, const short &level)
- const short & cubature_level () const

\[ \text{return cubatureLevel} \]

Protected Member Functions

- NonDCubature (Model &model)
  \textit{constructor}

- \texttt{~NonDCubature} ()
  \textit{destructor}

- void get_parameter_sets (const Model &model)
  \texttt{Returns one block of samples (ndim * num_samples).}

- void check_input ()
  \texttt{verify self-consistency of data}

- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
Private Attributes

- short cubatureLevelSpec
  
  *the user specification for the cubature level*

- short cubatureLevel
  
  *communicated through sampling_reset()*

8.77.1 Detailed Description

integrals over uncorrelated uniforms.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of moments. It employs Clenshaw-Curtis cubature for use with uniform density functions and integration bounds.

8.77.2 Constructor & Destructor Documentation

8.77.2.1 NonDCubature (Model & model, const short & level)

This alternate constructor is used for on-the-fly generation and evaluation of numerical cubature points.

8.77.2.2 NonDCubature (Model & model) [protected]

class constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate nond_cubature method specification.

8.77.3 Member Function Documentation

8.77.3.1 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.

The documentation for this class was generated from the following files:

- NonDCubature.H
- NonDCubature.C
8.78 NonDEvidence Class Reference

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

Inheritance diagram for NonDEvidence::

```
NonDEvidence
  |     |     |
  |     |     |
  |     |     |
Iterator
  |     |     |
  |     |     |
  |     |     |
Analyzer
  |     |     |
  |     |     |
  |     |     |
NonD
  |     |     |
  |     |     |
  |     |     |
NonDEvidence
```

Public Member Functions

- **NonDEvidence (Model &model)**
  
  *constructor*

- **~NonDEvidence ()**
  
  *destructor*

- **void quantify_uncertainty ()**
  
  *for cumulative distribution functions of belief and plausibility*

- **void print_results (ostream &s) const**
  
  *print the cumulative distribution functions for belief and plausibility*

Protected Member Functions

- **void initialize_final_statistics ()**
  
  *initialize finalStatistics for belief/plausibility results sets*

- **void compute_statistics ()**
  
  *or vice-versa*
Private Member Functions

- void calculate_basic_prob_intervals ()
  basic probability assignments for input interval combinations

- void calculate_maxmin_per_interval (const size_t &func_num)
  maximum and minimum values within each input interval combination (cell).

- void calculate_cum_belief_plaus (const size_t &func_num)
  per interval cell

Private Attributes

- Iterator lhsSampler
  the LHS sampler instance

- const int originalSeed
  the user seed specification (default is 0)

- int numSamples
  the number of samples used in the surrogate

- int NV
  Size variable for DDS arrays.

- int NCMB
  Size variable for DDS arrays.

- int MAXINTVLS
  Size variable for DDS arrays.

- Real Y
  Temporary output variable.

- Real * BPA
  Internal DDS array.

- Real * VMIN
  Internal DDS array.

- Real * VMAX
  Internal DDS array.

- Real * BPAC
  Internal DDS array.
8.78 NonDEvidence Class Reference

- **Real * CMIN**
  *Internal DDS Array.*

- **Real * CMAX**
  *Internal DDS Array.*

- **Real * X**
  *Internal DDS Array.*

- **int * NI**
  *Internal DDS array.*

- **int * IP**
  *Internal DDS array.*

- **int * IPBEL**
  *Internal DDS array.*

- **int * IPPLA**
  *Internal DDS array.*

- **RealVectorArray cc_bel_fn**
  *Storage array to hold CCBF values.*

- **RealVectorArray cc_plaus_fn**
  *Storage array to hold CCPF values.*

- **RealVectorArray cc_bel_val**
  *Storage array to hold CCB response values.*

- **RealVectorArray cc_plaus_val**
  *Storage array to hold CCP response values.*

- **VariablesArray all_vars**
  *Storage array to hold variables.*

- **ResponseArray all_responses**
  *Storage array to hold responses.*

### 8.78.1 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.

8.78.2 Member Data Documentation

8.78.2.1 int NV [private]
Size variable for DDS arrays.
NV = number of interval variables

8.78.2.2 int NCMB [private]
Size variable for DDS arrays.
NCMB = number of cell combinations

8.78.2.3 int MAXINTVLS [private]
Size variable for DDS arrays.
MAXINTVLS = maximum number of intervals per individual interval var

8.78.2.4 Real Y [private]
Temporary output variable.
Y = current output to be placed in cell

8.78.2.5 Real BPA [private]
Internal DDS array.
Basic Probability Assignments

8.78.2.6 Real VMIN [private]
Internal DDS array.
Minimum ends of intervals.
8.78.2.7  **Real** VMAX  [private]
Internal DDS array.
Maximum ends of intervals.

8.78.2.8  **Real** BPAC  [private]
Internal DDS array.
Basic Probability Combinations.

8.78.2.9  **Real** CMIN  [private]
Internal DDS Array.
Minimum per cell combination.

8.78.2.10 **Real** CMAX  [private]
Internal DDS Array.
Maximum per cell combination.

8.78.2.11 **Real** X  [private]
Internal DDS Array.
X per cell combination.

8.78.2.12  **int** NI  [private]
Internal DDS array.
Number of intervals per interval variable

8.78.2.13  **int** IP  [private]
Internal DDS array.
Sort order for combinations

8.78.2.14  **int** IPBEL  [private]
Internal DDS array.
Sort order for belief values
8.78.2.15 int *IPPLA [private]

Internal DDS array.
Sort order for belief values

The documentation for this class was generated from the following files:

- NonDEvidence.H
- NonDEvidence.C
8.79 NonDGlobalReliability Class Reference

Class for global reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDGlobalReliability:

```
   NonDGlobalReliability
      |               
      v               
   NonDReliability
      |               
      v               
   NonD
      |               
      v               
   Analyzer
      |               
      v               
   Iterator
```

Public Member Functions

- **NonDGlobalReliability** (Model &model)
  
  *constructor*

- **~NonDGlobalReliability** ()
  
  *destructor*

- **void quantify_uncertainty** ()
  
  *approximations of the cumulative distribution function of response*

- **void print_results** (ostream &s) const
  
  *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** ()
importance 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 RealVector &c_variables)
  expected improvement function for the GP

- Real `expected_feasibility` (const RealVector &expected_values, const RealVector &c_variables)
  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)
  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)
  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`
  current iterate should be accepted (must reduce violation)

- bool `lastIterateAccepted`
  this controls update of parameters for augmented Lagrangian merit function
Static Private Attributes

- static NonDGlobalReliability * nondGlobRelInstance

functions in order to avoid the need for static data

8.79.1 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.H
- NonDGlobalReliability.C
8.80 NonDIncremLHSSampling Class Reference

Performs incremental LHS sampling for uncertainty quantification.

Inheritance diagram for NonDIncremLHSSampling::

```
NonDIncremLHSSampling
  NonDSampling
    NonD
      Analyzer
        Iterator
```

Public Member Functions

- `NonDIncremLHSSampling (Model &model)`
  
  constructor

- `~NonDIncremLHSSampling ()`

  destructor

- void `quantify_uncertainty ()`

  parameter samples, and computing statistics on the ensemble of results.

- void `print_results (ostream &s) const`

  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

- bool varBasedDecompFlag
  flags computation of VBD

Static Private Attributes

- static RealArray rawData
  vector to hold raw data before rank sort

8.80.1 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.

8.80.2 Constructor & Destructor Documentation

8.80.2.1 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.

8.80.3 Member Function Documentation

8.80.3.1 void quantify_uncertainty () [virtual]

parameter samples, and computing statistics on the ensemble of results.
Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.
Implements NonD.

The documentation for this class was generated from the following files:

- NonDIncremLHSSampling.H
- NonDIncremLHSSampling.C
8.81 NonDIntegration Class Reference

numerical integration points for evaluation of expectation integrals

Inheritance diagram for NonDIntegration::

```
    Iterator
     |
     v
Analyzer
     |
     v
NonD
     |
     v
NonDIntegration
     |
     v
NonDCubature
     |
     v
NonDQuadrature
```

Public Member Functions

- const RealVector & weight_products () const
  
  return weightProducts

Protected Member Functions

- NonDIntegration (Model &model)
  
  constructor

- NonDIntegration (NoDBBaseConstructor, Model &model)
  
  alternate constructor for instantiations "on the fly"

- ~NonDIntegration ()
  
  destructor

- virtual void check_input ()=0
  
  verify self-consistency of data

- void quantify_uncertainty ()
  
  distributions into response statistics
Protected Attributes

- `RealVector weightProducts`
  
  *n-dimensional stencil*

Private Attributes

- `size_t numIntegrations`
  
  *counter for number of integration executions for this object*

8.81.1 Detailed Description

Numerical integration points for evaluation of expectation integrals

This class provides a base class for shared code among `NonDQuadrature` and `NonDCubature`.

8.81.2 Constructor & Destructor Documentation

8.81.2.1 `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_cubature` method specifications.

8.81.2.2 `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.

The documentation for this class was generated from the following files:

- `NonDIntegration.H`
- `NonDIntegration.C`
8.82 NonDLHSSampling Class Reference

Performs LHS and Monte Carlo sampling for uncertainty quantification.

Inheritance diagram for NonDLHSSampling::

```
  Iterator
   
  Analyzer
   
  NonD
   
  NonDSampling
   
  NonDLHSSampling
```

Public Member Functions

- `NonDLHSSampling (Model &model)`
  standard constructor

- `NonDLHSSampling (Model &model, int samples, int seed, short sampling_vars_mode=ACTIVE)`
  alternate constructor for sample generation and evaluation "on the fly"

- `NonDLHSSampling (int samples, int seed, const RealVector &lower_bnds, const RealVector &upper_bnds)`
  alternate constructor for sample generation "on the fly"

- `~NonDLHSSampling ()`
  destructor

Protected Member Functions

- `void quantify_uncertainty ()`
  parameter samples, and computing statistics on the ensemble of results.

- `void print_results (ostream &s) const`
  print the final statistics
Private Attributes

- bool varBasedDecompFlag
  
  flags computation of VBD

8.82.1 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.

8.82.2 Constructor & Destructor Documentation

8.82.2.1 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.

8.82.2.2 NonDLHSSampling (Model & model, int samples, int seed, short sampling_vars_mode = ACTIVE)

alternate constructor for sample generation and evaluation "on the fly"

This alternate constructor is used by NonDEvidence for generation and evaluation of Model-based 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. It’s purpose is to avoid the need for a separate LHS specification within methods that use LHS sampling.

8.82.2.3 NonDLHSSampling (int samples, int seed, 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).
8.82.3 Member Function Documentation

8.82.3.1 void quantify_uncertainty () [protected, virtual]

parameter samples, and computing statistics on the ensemble of results.
Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.
Implements NonD.
The documentation for this class was generated from the following files:

- NonDLHSSampling.H
- NonDLHSSampling.C
8.83 NonDLocalReliability Class Reference

Class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDLocalReliability:

```
NonDLocalReliability
  NonDReliability
    NonD
      Analyzer
        Iterator
```

Public Member Functions

- `NonDLocalReliability (Model &model)`
  
  `constructor`

- `~NonDLocalReliability ()`
  
  `destructor`

- `void quantify_uncertainty ()`
  
  `approximations of the cumulative distribution function of response`

- `void print_results (ostream &s) const`
  
  `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 ()`
Taylor-series approximation.

- void mean_value ()
  computation of approximate statistics and importance factors

- void mpp_search ()
  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 ()
  data for each response function prior to level 0

- void initialize_mpp_search_data ()
  data for each z/p/beta level for each response function

- void update_mpp_search_data (const Variables &vars_star, const Response &resp_star)
  z/p/beta level for each response function

- void update_level_data (RealVector &final_stats, RealMatrix &final_stat_grads)
  statistics following MPP convergence

- void update_pma_reliability_level ()
  generalized reliabilities by inverting second-order integrations

- void update_limit_state_surrogate ()
  to the data fit embedded within uSpaceModel

- void assign_mean_data ()
  from ranVarMeansX/U, fnValsMeanX, fnGradsMeanX, and fnHessiansMeanX

- void dg_ds_eval (const Epetra_SerialDenseVector &x_vars, const Epetra_SerialDenseVector &fn_grad_x, RealMatrix &final_stat_grads)
  convenience function for evaluating dg/ds

- Real probability (const Real &beta, bool cdf_flag)
  second-order integration

- Real reliability (const Real &p, bool cdf_flag)
  second-order integration

- bool reliability_residual (const Real &p, const Real &beta, const Epetra_SerialDenseVector &kappa, Real &res)
  corrections using Newton's method (called by reliability(p))
- Real `reliability_residual_derivative` (const Real &p, const Real &beta, const Epetra_SerialDenseVector &kappa)
  probability corrections using Newton's method (called by reliability(p))

- void `principal_curvatures` ()
  Compute the kappaU vector of principal curvatures from fnHessU.

**Private Attributes**

- Epetra_SerialDenseVector `fnGradX`
  evaluation

- Epetra_SerialDenseVector `fnGradU`
  Jacobian dx/du.

- Epetra_SerialSymDenseMatrix `fnHessX`
  evaluation

- Epetra_SerialSymDenseMatrix `fnHessU`
  Jacobian dx/du.

- Epetra_SerialDenseVector `kappaU`
  transformation of fnHessU

- Epetra_SerialDenseVector `fnValsMeanX`
  response function values evaluated at mean x

- Epetra_SerialDenseMatrix `fnGradsMeanX`
  response function gradients evaluated at mean x

- Array < Epetra_SerialSymDenseMatrix > `fnHessiansMeanX`
  response function Hessians evaluated at mean x

- RealVector `medianFnVals`
  \( p=0.5 \rightarrow \text{median function values} \). Used to determine the sign of beta.

- Epetra_SerialDenseVector `ranVarMeansU`
  vector of means for all uncertain random variables in u-space

- RealVector `initialPtU`
  initial guess for MPP search in u-space

- Epetra_SerialDenseVector `mostProbPointX`
  location of MPP in x-space
- `Epetra_SerialDenseVector mostProbPointU
  location of MPP in u-space`

- `RealVectorArray prevMPPULev0
  initialPtU within RBDO.

- `RealMatrix prevFnGradDL Lev0
  for level 0. Used for warm-starting initialPtU within RBDO.

- `RealMatrix prevFnGradULev0
  for level 0. Used for warm-starting initialPtU within RBDO.

- `RealVector prevICVars
  previous design vector. Used for warm-starting initialPtU within RBDO.

- `ShortArray prevCumASVLev0
  for warm-starting initialPtU within RBDO.

- `bool npsolFlag
  selection (SQP or NIP)

- `bool warmStartFlag
  flag indicating the use of warm starts

- `bool nipModeOverrideFlag
  flag indicating the use of move overrides within OPT++ NIP

- `bool curvatureDataAvailable
  mostProbPointU) is available for computing 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
  derived from hessianType

- `RealMatrix impFactor
  importance factors predicted by MV

- `int npsolDerivLevel

DAKOTA Version 4.1 Developers Manual generated on October 11, 2007
fn, 2 = analytic grads of constraints, 3 = analytic grads of both).

- unsigned short warningBits
  set of warnings accumulated during execution

## 8.83.1 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 (MVFOSSM/MVSOSM), 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.

## 8.83.2 Member Function Documentation

### 8.83.2.1 void initial_taylor_series () [private]

Taylor-series approximation.

An initial first- or second-order Taylor-series approximation is required for MV/AMV/AMV+/TANA or for the case where meanStats or stdDevStats (from MV) are required within finalStatistics for subIterator usage of NonDLocalReliability.

### 8.83.2.2 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.

### 8.83.2.3 void initialize_level_data () [private]

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).
8.83.2.4 void initialize_mpp_search_data () [private]

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).

8.83.2.5 void update_mpp_search_data (const Variables & vars_star, const Response & resp_star) [private]

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.

8.83.2.6 void update_level_data (RealVector & final_stats, RealMatrix & final_stat_grads) [private]

statistics following MPP convergence
Updates computedRespLevels/computedProbLevels/computedRelLevels, final_stats/final_stat_grads, warm start, and graphics data.

8.83.2.7 void update_pma_reliability_level () [private, virtual]

generalized reliabilities by inverting second-order integrations
For PMA SORM with prescribed p-level or prescribed generalized beta-level, requestedCDFRelLevel must be updated. This virtual function redefinition is called from NonDReliability::PMA_constraint_eval().
Reimplemented from NonDReliability.

8.83.2.8 void dg_ds_eval (const Epetra_SerialDenseVector & x_vars, const Epetra_SerialDenseVector & fn_grad_x, RealMatrix & final_stat_grads) [private]

convenience function for evaluating dg/ds
Computes dg/ds where s = design variables. Supports potentially overlapping cases of design variable augmentation and insertion.

8.83.2.9 Real probability (const Real & beta, bool cdf_flag) [private]

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).
8.83.2.10  Real reliability (const Real & p, bool cdf_flag)  [private]

second-order integration

Converts a probability into a reliability using the inverse of the first-order or second-order integrations implemented in NonDLocalReliability::probability().

The documentation for this class was generated from the following files:

- NonDLocalReliability.H
- NonDLocalReliability.C
8.84 NonDPolynomialChaos Class Reference

quantification

Inheritance diagram for NonDPolynomialChaos:

```
  +-------------+                +-------------+                +-------------+
  | Iterator    |                | Analyzer    |                | NonD         |
  |             |                |             |                |             | NonDPolynomialChaos
```

**Public Member Functions**

- **NonDPolynomialChaos (Model &model)**  
  constructor

- ```~NonDPolynomialChaos ()```  
  destructor

- ```void quantify_uncertainty ()```  
  perform a forward uncertainty propagation using SFEM/PCE methods

- ```void print_results (ostream &s) const```  
  print the final statistics and PCE coefficient array

**Private Attributes**

- **Model uSpaceModel**  
  u-space recasting and orthogonal polynomial data fit recursions

- **Iterator pceSampler**  
  an LHS sampling instance, but AIS could also be used.

- ```size_t numPCEAnalyses```  
  number of invocations of `quantify_uncertainty()`
8.84.1 Detailed Description

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.

The documentation for this class was generated from the following files:

- NonDPolynomialChaos.H
- NonDPolynomialChaos.C
8.85 NonDQuadrature Class Reference

normals/uniforms/exponentials/betas/gammas.
Inheritance diagram for NonDQuadrature::

```
       Iteration
        ^       
        |       
        |       
        v       
  NonDIntegration
        ^       
        |       
        |       
        v       
  NonD
        ^       
        |       
        |       
        v       
  Analyzer
        ^       
        |       
        |       
        v       
  Iterator
```

**Public Member Functions**

- `NonDQuadrature (Model &model, const ShortArray &order)`
- `const ShortArray & quadrature_order ()` const
  
  ```
  return quadOrder
  ```

**Protected Member Functions**

- `NonDQuadrature (Model &model)`
  
  ```
  constructor
  ```

- `~NonDQuadrature ()`
  
  ```
  destructor
  ```

- `void get_parameter_sets (const Model &model)`
  
  ```
  Returns one block of samples (ndim * num_samples).
  ```

- `void check_input ()`
  
  ```
  verify self-consistency of data
  ```

- `void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)`
Private Attributes

- ShortArray quadOrderSpec
  
  the user specification for the number of Gauss points per dimension

- ShortArray quadOrder
  
  external requirements communicated through sampling_reset()

8.85.1 Detailed Description

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.

8.85.2 Constructor & Destructor Documentation

8.85.2.1 NonDQuadrature (Model & model, const ShortArray & order)

This alternate constructor is used for on-the-fly generation and evaluation of numerical quadrature points.

8.85.2.2 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.

8.85.3 Member Function Documentation

8.85.3.1 void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag) [inline, 
  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.
The documentation for this class was generated from the following files:

- NonDQuadrature.H
- NonDQuadrature.C
8.86 NonDReliability Class Reference

Base class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDReliability:

```
    Iterator
     |
    Analyzer
     |
        NonD
        |
    NonDReliability
     |
  NonDGlobalReliability
  NonDLocalReliability
```

Public Member Functions

- `NonDReliability (Model &model)`  
  constructor

- `~NonDReliability ()`  
  destructor

Protected Member Functions

- virtual void `update_pma_reliability_level ()`  
  update requestedCDFRelLevel for use in PMA_constraint_eval()

- void `jacobian_dX_dS` (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseMatrix &jacobian_, xs)  
  `trans_U_to_Xi()` with respect to distribution parameters S

- void `numerical_design_jacobian` (const Epetra_SerialDenseVector &x_vars, bool xs, Epetra_SerialDenseMatrix &num_jacobian_xs, bool zs, Epetra_SerialDenseMatrix &num_jacobian_zs)  
  and zs booleans

- void `verify_trans_jacobian_hessian` (const Epetra_SerialDenseVector &v0)  
  routine for verification of transformation Jacobian/Hessian terms
• void `verify_design_jacobian` (const Epetra_SerialDenseVector &u0)
  routine for verification of design Jacobian terms

• const Real & `distribution_parameter` (const size_t &index)
  return a particular random variable distribution parameter

• void `distribution_parameter` (const size_t &index, const Real &param)
  update derived quantities

Static Protected Member Functions

• static void `RIA_objective_eval` (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  (MPP) with the objective function of $(\text{norm } u)^2$.

• static void `RIA_constraint_eval` (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  (MPP) with the constraint of $G(u) = \text{response level}$.

• static void `PMA_objective_eval` (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  (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)
  (MPP) with the constraint of $(\text{norm } u)^2 = \beta^2$.

• static void `PMA2_asv_mapping` (const ShortArray &recast_asv, ShortArray &sub_model_asv)
  beta-bar constraint target update is required for second-order PMA

Protected Attributes

• Model uSpaceModel
  recastings and data fits

• Model mppModel
  `RecastModel` which formulates the optimization subproblem: RIA, PMA, EGO.

• Iterator mppOptimizer
  `Iterator` which optimizes the mppModel.

• short mppSearchType
- **Iterator importanceSampler**
  importance sampling instance used to compute/refine probabilities

- **short integrationRefinement**
  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

- **Real requestedRespLevel**
  the response level target for the current response function

- **Real requestedCDFProbLevel**
  the CDF probability level target for the current response function

- **Real requestedCDFRelLevel**
  the CDF reliability level target for the current response function

- **Real computedRespLevel**
  output response level calculated

- **Real computedRelLevel**
  output reliability level calculated

### Static Protected Attributes

- **static NonDReliability * nondRelInstance**
  functions in order to avoid the need for static data
8.86 NonDReliability Class Reference

8.86.1 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.

8.86.2 Member Function Documentation

8.86.2.1 void RIA_objective_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(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.

8.86.2.2 void RIA_constraint_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(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.

8.86.2.3 void PMA_objective_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(MPP) with the objective function of G(u).

This function recasts a G(u) response set (already transformed and approximated in other recursions) into a PMA objective function.

8.86.2.4 void PMA_constraint_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(MPP) with the constraint of (norm u)^2 = beta^2.

This function recasts a G(u) response set (already transformed and approximated in other recursions) into a PMA equality constraint.
8.86.2.5  void jacobian_dX_dS (const Epetra_SerialDenseVector & x_vars, Epetra_SerialDenseMatrix & jacobian_xs) [protected]

trans_U_to_X() with respect to distribution parameters S

This procedure computes the derivative of the original variables x with respect to the random variable distribution parameters s. This provides the design Jacobian of the transformation for use in computing RBDO design sensitivities.

8.86.2.6  void numerical_design_jacobian (const Epetra_SerialDenseVector & x_vars, bool xs, Epetra_SerialDenseMatrix & num_jacobian_xs, bool zs, Epetra_SerialDenseMatrix & num_jacobian_zs) [protected]

and zs booleans

This procedure computes numerical derivatives of x and/or z with respect to distribution parameters s, and is used by jacobian_dX_dS() to provide data that is not available analytically. Numerical dz/ds involves dL/ds (z(s) = L(s) u) and dz/ds = dL/ds u) and is needed to evaluate dx/ds semi-analytically for correlated variables. Numerical dx/ds is needed for distributions lacking simple closed-form CDF expressions (beta and gamma distributions).

The documentation for this class was generated from the following files:

- NonDReliability.H
- NonDReliability.C
8.87 NonDSampling Class Reference

NonDIncremLHSSampling, and NonDAdaptImpSampling.

Inheritance diagram for NonDSampling::

```
NonDSampling
    |     |
    |     |
    NonD
    |     |
    |     |
    NonDSampling
```

### Public Member Functions

- void compute_distribution_mappings (const ResponseArray &samples)
  - z to p/beta and of p/beta to z

- void compute_correlations (const VariablesArray &vars_samples, const ResponseArray &resp_samples)
  - simple, partial, simple rank, and partial rank

- void update_final_statistics ()
  - and computedProbLevels/computedRelLevels/computedRespLevels

- void print_distribution_mappings (ostream &s) const
  - prints the p/beta/z mappings computed in compute_distribution_mappings()

- void print_correlations (ostream &s) const
  - prints the correlations computed in compute_correlations()

### Protected Member Functions

- NonDSampling (Model &model)
  - constructor

- NonDSampling (NoDBBaseConstructor, Model &model, int samples, int seed)
alternate constructor for sample generation and evaluation "on the fly"

- **NonDSampling** (NoDBaseConstructor, int samples, int seed, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation "on the fly"

- ~NonDSampling ()
  destructor

- 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 (const Model &model)
  called several times.

- void get_parameter_sets (const RealVector &lower_bnds, const RealVector &upper_bnds)
  lower_bnds/upper_bnds.

- void run_lhs (const RealVector &d_l_bnds, const RealVector &d_u_bnds, const RealVector &s_l_bnds, const RealVector &s_u_bnds, const RealVector &n_means, const RealVector &n_std_devs, const RealVector &ln_l_bnds, const RealVector &ln_u_bnds, const RealVector &ln_err_facts, const RealVector &ln_l_bnds, const RealVector &ln_u_bnds, const RealVector &u_l_bnds, const RealVector &u_u_bnds, const RealVector &lu_l_bnds, const RealVector &lu_u_bnds, const RealVector &t_modes, const RealVector &t_l_bnds, const RealVector &t_u_bnds, const RealVector &b_alphas, const RealVector &b_betas, const RealVector &b_l_bnds, const RealVector &b_u_bnds, const RealVector &ga_alphas, const RealVector &ga_betas, const RealVector &w_alphas, const RealVector &w_betas, const RealVectorArray &h_bin_prs, const RealVectorArray &h_pt_prs, const RealVectorArray &i_probs, const RealVectorArray &i_bounds, const RealMatrix &correlations, RealVectorArray &rank_array, int num_samples, bool write_message)
  new LHS libraries.

- void compute_statistics (const VariablesArray &vars_samples, const ResponseArray &resp_samples)
  or intervals (epistemic or mixed uncertainties)

- void compute_intervals (const ResponseArray &samples)
  called by compute_statistics() to calculate min/max intervals

- void compute_moments (const ResponseArray &samples)
  deviations, and confidence intervals

- void print_statistics (ostream &s) const
prints the statistics computed in `compute_statistics()`

- `void print_intervals` (ostream &s) const
  prints the intervals computed in `compute_intervals()`

- `void print_moments` (ostream &s) const
  prints the moments computed in `compute_moments()`

- `void simple_corr` (Epetra_SerialDenseMatrix &total_data, bool rank_on, const int &num_in)
  computes simple correlations

- `void partial_corr` (Epetra_SerialDenseMatrix &total_data, bool rank_on, const int &num_in)
  computes partial correlations

**Static Protected Member Functions**

- `static bool rank_sort` (const int &x, const int &y)
  sort algorithm to compute ranks for rank correlations

**Protected Attributes**

- `const int originalSeed`
  the user seed specification (default is 0)

- `int samplesSpec`
  initial specification of number of samples

- `int numSamples`
  the current number of samples to evaluate

- `String sampleType`
  the sample type: random, lhs, or incremental_lhs

- `bool statsFlag`
  flags computation/output of statistics

- `bool allDataFlag`
  flags update of allVariables/allResponses

- `short samplingVarsMode`
  the sampling mode: ACTIVE, ACTIVE_UNIFORM, ALL, or ALL_UNIFORM

- `short sampleRanksMode`
SET_RANKS, or SET_GET_RANKS.

- bool varyPattern
  
  optimization) are not repeated, but are still repeatable

- RealVectorArray sampleRanks
  
  data structure to hold the sample ranks

- RealVector mean95CIDeltas
  
  intervals (calculated in compute_moments())

- RealVector stdDev95CILowerBnds
  
  (calculated in compute_moments())

- RealVector stdDev95CIUpperBnds
  
  (calculated in compute_moments())

Private Member Functions

- void check_error (const int &err_code, const char *err_source) const
  
  error is returned

Private Attributes

- int randomSeed
  
  the current random number seed

- size_t numLHSRuns
  
  counter for number of executions of run_lhs() for this object

- RealVector minValues
  
  (calculated in compute_intervals())

- RealVector maxValues
  
  (calculated in compute_intervals())

- Epetra_SerialDenseMatrix simpleCorr
  
  matrix to hold simple raw correlations

- Epetra_SerialDenseMatrix simpleRankCorr
  
  matrix to hold simple rank correlations

- Epetra_SerialDenseMatrix partialCorr
8.87 NonDSampling Class Reference

matrix to hold partial raw correlations

- Epetra_SerialDenseMatrix partialRankCorr
  matrix to hold partial rank correlations

Static Private Attributes

- static RealArray rawData
  vector to hold raw data before rank sort
- static int pgf90Initialized
  flag indicating whether pghpf_init() has been called.

8.87.1 Detailed Description

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 manages two LHS versions within a #ifdef construct in run_lhs(): (1) the 1998 Fortran 90 LHS version as documented in SAND98-0210, which was converted to a UNIX link library in 2001, (2) the 1970’s vintage LHS that had been f2c’d and converted to (incomplete) classes.

8.87.2 Constructor & Destructor Documentation

8.87.2.1 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.

8.87.2.2 NonDSampling (NoDBBaseConstructor, Model & model, int samples, int seed) [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.

8.87.2.3 NonDSampling (NoDBBaseConstructor, int samples, int seed, 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.

8.87.3 Member Function Documentation

8.87.3.1 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.

8.87.3.2 void get_parameter_sets (const Model & model) [protected, virtual]

called several times.
This version of get_parameter_sets() extracts data from the user-defined model in any of the four sampling modes.
Reimplemented from Analyzer.

8.87.3.3 void get_parameter_sets (const RealVector & lower_bnds, const RealVector & upper_bnds) [protected]

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.
The documentation for this class was generated from the following files:

- NonDSampling.H
- NonDSampling.C
8.88  NPSOLOptimizer Class Reference

Wrapper class for the NPSOL optimization library.

Inheritance diagram for NPSOLOptimizer::

```
NPSOLOptimizer
   |    |
   \
   |
Minimizer
   |    |
   \
   |
Optimizer
   |    |
   \
   |
SOLBase
```

### Public Member Functions

- **NPSOLOptimizer (Model &model)**  
  *standard constructor*

- **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 &, double *, int &), void(*user_con_eval)(int &, int &, int &, int &), const int &derivative_level, const Real &conv_tol)**  
  *alternate constructor for instantiations "on the fly"*

- **~NPSOLOptimizer ()**  
  *destructor*

- **void find_optimum ()**  
  *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)
  objective function (passed by function pointer to NPSOL).

Private Attributes

- String setUpType
  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 *, int &)
  "user_functions" mode.

- void(* userConstraintEval )(int &, int &, int &, int &, int *, double *, double *, double *, int &)
  "user_functions" mode.

Static Private Attributes

- static NPSOLOptimizer * npsolInstance
  functions in order to avoid the need for static data

8.88.1 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.

8.88.2 Constructor & Destructor Documentation

8.88.2.1 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.

8.88.2.2 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_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.

The documentation for this class was generated from the following files:

- NPSOLOptimizer.H
- NPSOLOptimizer.C
8.89 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy.

Inheritance diagram for Optimizer:

```
        Iterator
          ↓
        Minimizer
          ↓
        Optimizer
            ↓
   COLINOOptimizer
       ↓
CONMINOptimizer
       ↓
DOTOptimizer
       ↓
EffGlobalOptimizer
       ↓
JEGAOptimizer
       ↓
NCSUOptimizer
       ↓
NLPQLPOptimizer
       ↓
NPSOLOptimizer
       ↓
SNLLOptimizer
```

Public Member Functions

- void run ()
  
  run the iterator; portion of run_iterator()

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_dv, 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 **print_results** (ostream &s) const

- void **multi_objective_weights** (const RealVector &multi_obj_wts)
  - *Used by ConcurrentStrategy for Pareto set optimization.*

- void **derived_initialize_scaling** (StringArray &fn_scale_types, RealVector &fn_scales)
  - *respectively*

- void **derived_post_run** ()

- virtual void **find_optimum** ()=0
  - *Redefines the run virtual function for the optimizer branch.*

- **Response** **multi_objective_modify** (const Response &raw_response) const
  - *objective for single-objective optimizers*

- const **RealVector** & **multi_objective_retrieve** (const Variables &vars, const Response &response) const
  - *from the solution of a single-objective optimizer*

### Static Protected Member Functions

- static void **primary_resp_recast** (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &scaled_response)
  - *from native (user) to iterator space*

### Protected Attributes

- **size_t** numObjectiveFunctions
  - *number of objective functions (iterator view)*
- `size_t numUserObjectiveFunctions`
  
  number of objective functions (user’s model view)

- `std::vector<Real> multiObjWeights`
  
  user-specified weights for multiple objective functions

- `bool multiObjFlag`
  
  flag indicating whether multi-objective transformations are necessary

### Static Protected Attributes

- `static `Optimizer * optimizerInstance`
  
  pointer to `Optimizer` instance used in static member functions

#### 8.89.1 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`.

#### 8.89.2 Constructor & Destructor Documentation

##### 8.89.2.1 `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.

#### 8.89.3 Member Function Documentation

##### 8.89.3.1 `void run ()` [inline, virtual]

run the iterator; portion of `run_iterator()`

`Iterator` supports a construct/pre-run/run/post-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from `Iterator`
8.89 Optimizer Class Reference

8.89.3.2 void print_results (ostream & s) const [protected, virtual]
Redefines default iterator results printing to include optimization results (objective functions and constraints).
Reimplemented from Iterator.

8.89.3.3 void derived_post_run () [protected, virtual]
Implements portions of post_run specific to Optimizers.
Reimplemented from Iterator.

8.89.3.4 void primary_resp_recast (const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response) [static, protected]
from native (user) to iterator space
Objective function map from user/native space to iterator/scaled/combined space using a RecastModel. If resizing the response, copies the constraint (secondary) data from native_response too.

8.89.3.5 Response multi_objective_modify (const Response & raw_response) const [protected]
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 multiObjWeights) is the only technique supported currently. The weightings are used to scale function values, gradients, and Hessians as needed.

8.89.3.6 const RealVector & multi_objective_retrieve (const Variables & vars, const Response & response) const [protected]
from the solution of a single-objective optimizer
Retrieve a full multiobjective response based on the data returned by a single objective optimizer by performing a data_pairs search.
The documentation for this class was generated from the following files:
  - DakotaOptimizer.H
  - DakotaOptimizer.C
8.90 OrthogonalPolynomial Class Reference

Base class for the orthogonal polynomial class hierarchy.

Inheritance diagram for OrthogonalPolynomial:

```
OrthogonalPolynomial
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GenLaguerreOrthogPolynomial</td>
</tr>
<tr>
<td>JacobiOrthogPolynomial</td>
</tr>
<tr>
<td>LegendreOrthogPolynomial</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- `OrthogonalPolynomial ()`
  default constructor

- `OrthogonalPolynomial (short poly_type)`
  alternate constructor

- `OrthogonalPolynomial (const OrthogonalPolynomial &polynomial)`
  copy constructor

- virtual `~OrthogonalPolynomial ()`
  destructor

- `OrthogonalPolynomial operator= (const OrthogonalPolynomial &polynomial)`
  assignment operator

- virtual const Real & `get_value (const Real &x, size_t n)`
  retrieve the orthogonal polynomial value for a given parameter value

- virtual const Real & `get_gradient (const Real &x, size_t n)`
  retrieve the orthogonal polynomial gradient for a given parameter value

- virtual const Real & `norm_squared (size_t n)`
  of the orthogonality statement for the derived polynomial type.

- virtual const RealVector & `gauss_points (size_t n)`
  return the `gaussPoints` corresponding to polynomial order n

- virtual const RealVector & `gauss_weights (size_t n)`
  return the `gaussWeights` corresponding to polynomial order n

- virtual void `alpha_stat (const Real &alpha)`
set alphaPoly from alpha_stat

- virtual void beta_stat (const Real &beta)
  set betaPoly from beta_stat

- void gauss_check (size_t n)
  perform unit testing on the Gauss points/weights

Protected Member Functions

- orthogonalPolynomial (BaseConstructor)
  derived class constructors - Coplien, p. 139

- size_t factorial (size_t n)
  compute n!

- Real factorial_ratio (size_t num, size_t den)
  compute num!/den!

- size_t n_choose_k (size_t n, size_t k)
  compute n!/(k!(n-k)!

- Real pochhammer (const Real &m, size_t n)
  compute the Pochhammer symbol (m)_n = m*(m+1)...*(m+n-1)

Protected Attributes

- Real orthogPolyValue
  value of the 1-D orthogonal polynomial returned by get_value()

- Real orthogPolyGradient
  gradient of the 1-D orthogonal polynomial returned by get_gradient()

- Real orthogPolyNormSq
  <Poly_n, Poly_n> = ||Poly_n||^2 (returned by norm_squared())

- RealVector gaussPoints
  (x parameter values for which Poly_n(x) = 0)

- RealVector gaussWeights
  Gauss weights for one-dimensional Gaussian quadrature.
Private Member Functions

- OrthogonalPolynomial * get_polynomial (short poly_type)
  appropriate derived type.

Private Attributes

- OrthogonalPolynomial * polyRep
  pointer to the letter (initialized only for the envelope)
- int referenceCount
  number of objects sharing polyRep

8.90.1 Detailed Description

Base class for the orthogonal polynomial class hierarchy.

The OrthogonalPolynomial class is the base class for the univariate orthogonal polynomial class hierarchy in DAKOTA. One instance of an OrthogonalPolynomial is created for each variable within a multidimensional orthogonal polynomial basis function (a vector of OrthogonalPolynomials is contained in OrthogPolyApproximation, which may be mixed and matched in, e.g., the Wiener-Askey scheme for polynomial chaos). For memory efficiency and enhanced polymorphism, the orthogonal polynomial hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (OrthogonalPolynomial) serves as the envelope and one of the derived classes (selected in OrthogonalPolynomial::get_polynomial()) serves as the letter.

8.90.2 Constructor & Destructor Documentation

8.90.2.1 OrthogonalPolynomial ()

default constructor

The default constructor is used in Array<OrthogonalPolynomial> instantiations and by the alternate envelope constructor. polyRep is NULL in this case (problem_db is needed to build a meaningful instance). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

8.90.2.2 OrthogonalPolynomial (short poly_type)

alternate constructor

Envelope constructor which does not require access to problem_db. This constructor executes get_polynomial(type), which invokes the default constructor of the derived letter class, which in turn invokes the BaseConstructor of the base class.
8.90 OrthogonalPolynomial Class Reference

8.90.2.3 OrthogonalPolynomial (const OrthogonalPolynomial & polynomial)
copy constructor
Copy constructor manages sharing of polyRep and incrementing of referenceCount.

8.90.2.4 ~OrthogonalPolynomial () [virtual]
destructor
Destructor decrements referenceCount and only deletes polyRep when referenceCount reaches zero.

8.90.2.5 OrthogonalPolynomial (BaseConstructor) [protected]
derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all derived classes. get_polynomial() 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_polynomial() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~OrthogonalPolynomial).

8.90.3 Member Function Documentation

8.90.3.1 OrthogonalPolynomial operator= (const OrthogonalPolynomial & polynomial)
assignment operator

8.90.3.2 size_t factorial (size_t n) [inline, protected]
compute n!
This implementation is unprotected from overflow, but this should be fine for the polynomial orders that we would expect to encounter. Whenever possible, orthogonal polynomial implementations should use factorial_ratio() or n_choose_k() instead of factorial() to avoid size_t overflow.

8.90.3.3 Real factorial_ratio (size_t num, size_t den) [inline, protected]
compute num!/den!
This implementation sequences products in order to minimize the chances of overflow, and its use should be preferred to factorial() whenever possible.
8.90.3.4 **size_t n_choose_k (size_t n, size_t k)** [inline, protected]

compute n!/(k!(n-k)!) This implementation sequences products in order to minimize the chances of overflow, and its use should be preferred to factorial() whenever possible.

8.90.3.5 **Real pochhammer (const Real & m, size_t n)** [inline, protected]

compute the Pochhammer symbol (m)_n = m*(m+1)...*(m+n-1)

This is the rising/upper factorial formulation of the Pochhammer symbol (m)_n.

8.90.3.6 **OrthogonalPolynomial * get_polynomial (short poly_type)** [private]

appropriate derived type.

Used only by the envelope constructor to initialize polyRep to the appropriate derived type.

The documentation for this class was generated from the following files:

- OrthogonalPolynomial.H
- OrthogonalPolynomial.C
8.91 OrthogPolyApproximation Class Reference

approximation).

Inheritance diagram for OrthogPolyApproximation::

```
OrthogPolyApproximation

| Approximation
```

Public Member Functions

- **OrthogPolyApproximation ()**
  *default constructor*

- **OrthogPolyApproximation (ProblemDescDB &problem_db, const size_t &num_acv)**
  *standard constructor*

- **~OrthogPolyApproximation ()**
  *destructor*

- **void solution_approach (short soln_approach)**
  *set coeffSolnApproach*

- **short solution_approach () const**
  *get coeffSolnApproach*

- **void expansion_terms (const int &exp_terms)**
  *set numExpansionTerms*

- **const int & expansion_terms () const**
  *get numExpansionTerms*

- **void integration_iterator (const Iterator &iterator)**
  *set integrationRep*

- **void basis_types (const ShortArray &basis)**
  *set basisTypes*

- **const ShortArray & basis_types () const**
  *get basisTypes*
- void jacobi_alphas (const RealVector &alphas)
  pass alpha_stat parameters to JACOBI polynomial bases

- void jacobi_betas (const RealVector &betas)
  pass beta_stat parameters to JACOBI polynomial bases

- void generalized_laguerre_alphas (const RealVector &alphas)
  pass alpha_stat parameters to GENERALIZED_LAGUERRE polynomial bases

- void resolve_inputs ()
  (numExpansionTerms and approxOrder) based on user input

- const Real & get_mean ()
  return the mean of the PCE, treating all variables as random

- const Real & get_mean (const RealVector &x, const BoolDeque &random_vars_key)
  treating a subset of the variables as random

- const RealBaseVector & get_mean_gradient (const RealVector &x, const BoolDeque &random_vars_key, const IntArray &dvv)
  treating a subset of the variables as random

- const Real & get_variance ()
  return the variance of the PCE, treating all variables as random

- const Real & get_variance (const RealVector &x, const BoolDeque &random_vars_key)
  treating a subset of the variables as random

- const RealBaseVector & get_variance_gradient (const RealVector &x, const BoolDeque &random_vars_key, const IntArray &dvv)
  treating a subset of the variables as random

- const Real & norm_squared (size_t expansion_index)
  treating all variables as random

- const Real & norm_squared (size_t expansion_index, const BoolDeque &random_vars_key)
  treating a subset of the variables as random

Protected Member Functions

- int num_coefficients () const
  derived class approximation type in numVars dimensions

- int num_constraints () const
return the number of constraints to be enforced via anchorPoint

- const RealVector & approximation_coefficients () const
  return the coefficient array computed by find_coefficients()

- void approximation_coefficients (const RealVector &approx_coeffs)
  computing with find_coefficients()

- void find_coefficients ()
  orthogonal polynomials

- void print_coefficients (ostream &s) const
  print the coefficients for the expansion

- const Real & get_value (const RealVector &x)
  retrieve the response PCE value for a given parameter vector

- const RealBaseVector & get_gradient (const RealVector &x)
  and default DVV

- const RealBaseVector & get_gradient (const RealVector &x, const IntArray &dvv)
  and given DVV

**Private Member Functions**

- const RealVector & get_multivariate_polynomials (const RealVector &xi)
  evaluated at a particular parameter set

- void integration ()
  (chaosCoeffsSolnApproach is QUADRATURE or CUBATURE)

- void regression ()
  (chaosCoeffsSolnApproach is POINT_COLLOCATION)

- void expectation ()
  (chaosCoeffsSolnApproach is SAMPLING)

- void gradient_check ()
  cross-validates alternate gradient expressions
Private Attributes

- short chaosCoeffsSolnApproach
  
  \textit{QUADRATURE, CUBATURE, POINT\_COLLOCATION, or SAMPLING.}

- int numExpansionTerms
  
  number of terms in Polynomial Chaos expansion \(\text{length of chaosCoeffs}\)

- NonDIntegration * integrationRep
  
  weight products

- Array< OrthogonalPolynomial > polynomialBasis
  
  \((\text{wienerAskeyChaos}) \text{ based on multiIndex}\)

- ShortArray basisTypes
  
  \textit{HERMITE, LEGENDRE, LAGUERRE, JACOBI, or GENERALIZED\_LAGUERRE.}

- Sizet2DArray multiIndex
  
  \textit{of the multivariate orthogonal polynomials}

- RealVector wienerAskeyChaos
  
  a particular \(x_i\) (returned by get\_multivariate\_polynomials())

- Real pceMean
  
  expected value of the expansion

- RealVector pceMeanGradient
  
  gradient of the expected value of the expansion

- Real pceVariance
  
  variance of the expansion

- RealVector pceVarianceGradient
  
  gradient of the variance of the expansion

- Real multiPolyNormSq
  
  norm-squared of one of the multivariate polynomial basis functions

- RealVector chaosCoeffs
  
  \(\text{numExpansionTerms entries}\)

- RealVectorArray chaosSamples
  
  \(\text{numExpansionTerms} \times \text{num\_pts entries}\)
8.91.1 Detailed Description

The OrthogPolyApproximation class provides a global approximation based on orthogonal polynomials. It is used primarily for polynomial chaos expansions (for stochastic finite element approaches to uncertainty quantification).

8.91.2 Member Function Documentation

8.91.2.1 const Real & get_mean ()

return the mean of the PCE, treating all variables as random

In this case, all expansion variables are random variables and the mean of the expansion is simply the first chaos coefficient.

8.91.2.2 const Real & get_mean (const RealVector & x, const BoolDeque & random_vars_key)

returning a subset of the variables as random

In this case, a subset of the expansion variables are random variables and the mean of the expansion involves evaluating the expectation over this subset.

8.91.2.3 const RealBaseVector & get_mean_gradient (const RealVector & x, const BoolDeque & random_vars_key, const IntArray & dvv)

treating a subset of the variables as random

In this function, a subset of the expansion variables are random variables, for which the mean of the expansion is the expectation over the random subset and the derivative of the mean is the derivative of the remaining expansion over the non-random subset.

8.91.2.4 const Real & get_variance ()

return the variance of the PCE, treating all variables as random

In this case, all expansion variables are random variables and the variance of the expansion is the sum over all but the first term of the coefficients squared times the polynomial norms squared.

8.91.2.5 void integration () [private]

(chaosCoeffsSolnApproach is QUADRATURE or CUBATURE)

The coefficients of the PCE for the response are calculated using a Galerkin projection of the response against each multivariate orthogonal polynomial basis fn using the inner product ratio \( \langle f, \Psi \rangle / \langle \Psi^2 \rangle \), where inner product \( \langle a, b \rangle \) is the n-dimensional integral of \( a \ast b \ast \text{weighting over the support range of the n-dimensional (composite) }

DAKOTA Version 4.1 Developers Manual generated on October 11, 2007
weighting function. 1-D quadrature rules are defined for specific 1-D weighting functions and support ranges
and approximate the integral of \( f \) weighting as the Sum\_i \( w_i f_i \). To extend this to n-dimensions, a tensor
product quadrature rule or sparse grid cubature rule is applied using the product of 1-D weightings applied to the
n-dimensional stencil of points. It is not necessary to approximate the integral for the denominator numerically,
since this is available analytically.

8.91.2.6 void regression () [private]

(chaosCoeffsSolnApproach is POINT\_COLLOCATION)

In this case, regression is used in place of Galerkin projection. That is, instead of calculating the PCE coefficients
using inner product ratios, linear least squares is used to estimate the PCE coefficients which best match a set of
response samples. This approach is also known as stochastic response surfaces. The least squares estimation is
performed using DGELSS (SVD) or DGGLSE (equality-constrained) from LAPACK, based on the presence of
an anchorPoint.

8.91.2.7 void expectation () [private]

(chaosCoeffsSolnApproach is SAMPLING)

The coefficients of the PCE for the response are calculated using a Galerkin projection of the response against each
multivariate orthogonal polynomial basis \( \Psi \) using the inner product ratio \( \langle f, \Psi \rangle / \langle \Psi^2 \rangle \), where inner product
\( \langle a, b \rangle \) is the n-dimensional integral of \( a \)\( b \)\( \)weighting over the support range of the n-dimensional (composite)
weighting function. When interpreting the weighting function as a probability density function, \( \langle a, b \rangle \) = expected
value of \( a \)\( b \), which can be evaluated by sampling from the probability density function and computing the mean
statistic. It is not necessary to compute the mean statistic for the denominator, since this is available analytically.

8.91.2.8 void gradient_check () [private]

cross-validates alternate gradient expressions

This test works in combination with DEBUG settings in (Legendre,Laguerre,Jacobi,GenLaguerre)Orthog-
Polynomial::get\_gradient().

The documentation for this class was generated from the following files:

- OrthogPolyApproximation.H
- OrthogPolyApproximation.C
8.92  ParallelConfiguration Class Reference

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
   improves modularity by avoiding explicit usage of MPI_COMM_WORLD)

ParLevLIter siPLIter
   (there may be more than one per parallel configuration instance)

ParLevLIter iePLIter
   (there can only be one)

ParLevLIter eaPLIter
   (there can only be one)

Friends

   class ParallelLibrary
      streamline implementation

8.92.1 Detailed Description

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.H
8.93 ParallelLevel Class Reference

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 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

• 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
  streamline implementation

8.93.1 Detailed Description

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.H
8.94 ParallelLibrary Class Reference

message passing within these levels.

Public Member Functions

- **ParallelLibrary** (int &argc, char **&argv)
  
  *stand-alone mode constructor*

- **ParallelLibrary** ()
  
  *library mode constructor*

- **ParallelLibrary** (int 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 **String** &default_config, const **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 **String** &default_config, const **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 **String** &default_config, const **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)
  inputs (normal mode)

- void specify_outputs_restart (const char *clh_std_output_filename, const char *clh_std_error_filename, const char *clh_read_restart_filename, const char *clh_write_restart_filename, int restart_evals)
  inputs (library mode).

- void manage_outputs_restart (const ParallelLevel &pl)
  manage output streams and restart file(s) (both modes)

- void close_streams ()
  close streams, files, and any other services

- 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_ag, 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)
  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
  
  parallel level

• bool si_parallel_level_defined () const
  
  strategy-iterator parallel level

• bool ie_parallel_level_defined () const
  
  iterator-evaluation parallel level

• bool ea_parallel_level_defined () const
  
  evaluation-analysis parallel level

• Array < MPI_Comm > analysis_intra_communicators ()
  
  prior to execution time).
Private Member Functions

- 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 String &default_config, const 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, const int &proc_remainder)
  
  *and num_servers child communicators*

- bool split_communicator_peer_partition (const ParallelLevel &parent_pl, ParallelLevel &child_pl, const int &proc_remainder)
  
  *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 String &default_cong, const 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)
  
  *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 String &err_source, const int &err_code)
  
  check the MPI return code and abort if error

**Private Attributes**

- ofstream output_ofstream
  
  tagged file redirection of stdout

- ofstream error_ofstream
  
  tagged file redirection of stderr

- int worldRank
  
  rank in MPI_COMM_WORLD

- int worldSize
  
  size of MPI_COMM_WORLD

- 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 stdOutputFlag
  flags redirection of DAKOTA std output to a file

- bool stdErrorFlag
  flags redirection of DAKOTA std error to a file

- 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

- const char * stdOutputFilename
  filename for redirection of stdout

- const char * stdErrorFilename
  filename for redirection of stderr

- const char * readRestartFilename
  input filename for restart

- const char * writeRestartFilename
  output filename for restart

- int restartEvals
  number of restart evals to read

- List< ParallelLevel > parallelLevels
  parallelism among one or more configurations

- List< ParallelConfiguration > parallelConfigurations
  indexing into parallelLevels
8.94.1 Detailed Description

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.

8.94.2 Constructor & Destructor Documentation

8.94.2.1 ParallelLibrary (int & argc, char **& argv)

stand-alone mode constructor

This constructor is the one used by main.C. It calls MPI_Init conditionally based on whether a parallel launch is detected.

8.94.2.2 ParallelLibrary ()

library mode constructor

This constructor provides a library mode and is used by the SIERRA Adak application. It does not call MPI_Init, but rather gathers data from MPI_COMM_WORLD if MPI_Init has been called elsewhere.

8.94.2.3 ParallelLibrary (int 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.

8.94.3 Member Function Documentation

- ParLevLIter currPLIter
  
  list iterator identifying the current node in parallelLevels

- ParConfigLIter currPCIter
  
  list iterator identifying the current node in parallelConfigurations
8.94.3.1  void specify_outputs_restart (CommandLineHandler & cmd_line_handler)

inputs (normal mode)
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 and 0 for restart_evals if no user specification. Only worldRank==0 has access to command line arguments and must Bcast this data to all iterator masters.

8.94.3.2  void manage_outputs_restart (const ParallelLevel & pl)

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.

8.94.3.3  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.

8.94.3.4  void increment_parallel_configuration ()  [inline]

add a new node to parallelConfigurations and increment currPCLiter
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).

8.94.3.5  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 String & default_config, const 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.
8.94.3.6  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 String & default_config, const 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.

The documentation for this class was generated from the following files:

- ParallelLibrary.H
- ParallelLibrary.C
8.95  ParamResponsePair Class Reference

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 (istream &s)
  read a ParamResponsePair object from an istream

- void write (ostream &s) const
  write a ParamResponsePair object to an ostream

- void read_annotated (istream &s)
  read a ParamResponsePair object in annotated format from an istream

- void write_annotated (ostream &s) const
  write a ParamResponsePair object in annotated format to an ostream

- void write_tabular (ostream &s) const
  write a ParamResponsePair object in tabular format to an ostream

- void read (BiStream &s)
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 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

const String & interface_id () const
return the interface identifier from the response object

Private Attributes

- Variables prPairParameters
  the set of parameters for the function evaluation

- Response prPairResponse
  the response set for the function evaluation

- String idInterface
  detection on results from different interfaces.

- int evalId
  ApplicationInterface::fnEvalId).
8.95 ParamResponsePair Class Reference

Friends

- bool operator==(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  equality operator

- bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  inequality operator

8.95.1 Detailed Description

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 list, in restart file operations, and in a variety of scheduling algorithm bookkeeping operations. With the advent of STL, replacement of arrays of this class with map<> and pair<> template constructs may be possible (using map<int, pair<vars,response>>, for example), assuming that deep copies, I/O, alternate constructors, etc., can be adequately addressed.

8.95.2 Constructor & Destructor Documentation

8.95.2.1 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).

8.95.2.2 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., beforeSynchCorePRPList, data_pairs) are involved.

8.95.3 Member Function Documentation
8.95.3.1  void read (MPIUnpackBuffer & s) [inline]

read a ParamResponsePair object from a packed MPI buffer
idInterface is omitted since master processor retains interface ids and communicates asv and response data only
with slaves.

8.95.3.2  void write (MPIPackBuffer & s) const [inline]

write a ParamResponsePair object to a packed MPI buffer
idInterface is omitted since master processor retains interface ids and communicates asv and response data only
with slaves.

8.95.4  Member Data Documentation

8.95.4.1  String idInterface [private]

detection on results from different interfaces.
idInterface 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().

8.95.4.2  int evalId [private]

ApplicationInterface::fnEvalId).
evalId belongs here rather than in Response since some Response objects involve consolidation of several fn evals
(e.g., Model::synchronize_derivatives()). The prPair, on the other hand, is used for storage of all low level fn evals
that get evaluated in ApplicationInterface::map().

The documentation for this class was generated from the following files:

- ParamResponsePair.H
- ParamResponsePair.C
8.96 ParamStudy Class Reference

Class for vector, list, centered, and multidimensional parameter studies.

Inheritance diagram for ParamStudy::

```
Public Member Functions

- ParamStudy (Model &model)
  constructor

- ~ParamStudy ()
  destructor

- void extract_trends ()
  Redefines the run_iterator virtual function for the PStudy/DACE branch.
```

```
Private Member Functions

- void compute_vector_steps ()
  and either numSteps or stepLength (pStudyType is 1 or 2)

- void sample (const RealVectorArray &list_of_points)
  performs the parameter study by sampling from a list of points

- void vector_loop (const RealVector &start, const RealVector &step_vect, const int &num_steps)
  increments of step_vect. Total number of evaluations is num_steps + 1.

- void centered_loop (const RealVector &start, const Real &percent_delta, const int &deltas_per_variable)
  centered about start
```
void multidim_loop (const IntArray &var_partitions)

performs vector_loops recursively in multiple dimensions

Private Attributes

- RealVectorArray listOfPoints
  array of evaluation points for the list_parameter_study

- RealVector initialPoint
  the starting point for vector and centered parameter studies

- RealVector finalPoint
  the ending point for vector_parameter_study (a specification option)

- RealVector stepVector
  the n-dimensional increment in vector_parameter_study

- int numSteps
  the number of times stepVector is applied in vector_parameter_study

- short pStudyType
  (different vector specifications), 4 (centered), or 5 (multidim)

- int deltasPerVariable
  variable in a centered_parameter_study

- Real stepLength
  (a specification option)

- Real percentDelta
  centered_parameter_study

- IntArray variablePartitions
  number of partitions for each variable in a multidim_parameter_study

8.96.1 Detailed Description

Class for vector, list, centered, and multidimensional parameter studies.

The ParamStudy class contains several algorithms for performing parameter studies of different types. It is not a wrapper for an external library, rather its algorithms are self-contained. 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 a specified number.
of intervals for each dimension. It is a nested study in that it utilizes the vector parameter study internally as it
recurses through the variables. 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.

The documentation for this class was generated from the following files:

- ParamStudy.H
- ParamStudy.C
8.97 ProblemDescDB Class Reference

The database containing information parsed from the DAKOTA input file.

Inheritance diagram for ProblemDescDB:

- ProblemDescDB
- IDRProblemDescDB

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)**
  *passed with the "-input" option on the DAKOTA command line.*

- **void manage_inputs (const char *dakota_input_file)**
  *This version reads from the dakota input filename passed in.*

- **void check_input ()**
  *keywords in the dakota input file. Used by manage_inputs().*

- **void lock ()**
  *may not be set properly. Unlocked by a set nodes operation.*

- **void set_db_list_nodes (const String &method_tag)**
  *this method specification to set all other list iterators.*
- void **set_db_list_nodes** (const size_t &method_index) 
  specification to set all other list iterators.

- void **set_db_method_node** (const size_t &method_index) 
  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) 
  identifier string

- void **set_db_model_nodes** (const size_t &model_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 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

- `const RealVector & getDrv (const String &entry_name) const`
  get a `RealVector` out of the database based on an identifier string

- `const IntVector & getDiv (const String &entry_name) const`
  get an `IntVector` out of the database based on an identifier string

- `const IntArray & getDia (const String &entry_name) const`
  get an `IntArray` out of the database based on an identifier string

- `const ShortArray & getDsha (const String &entry_name) const`
  get a `ShortArray` out of the database based on an identifier string

- `const RealMatrix & getDrm (const String &entry_name) const`
  get a `RealMatrix` out of the database based on an identifier string

- `const RealVectorArray & getDrva (const String &entry_name) const`
  get a `RealVectorArray` out of the database based on an identifier string

- `const IntList & getDil (const String &entry_name) const`
  get an `IntList` out of the database based on an identifier string

- `const IntSet & getDis (const String &entry_name) const`
  get an `IntSet` out of the database based on an identifier string

- `const StringArray & getDsa (const String &entry_name) const`
  get a `StringArray` out of the database based on an identifier string

- `const String2DArray & getDs2a (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

- const int & get_int (const String &entry_name) const
  get an int out of the database based on an identifier string

- const short & get_short (const String &entry_name) const
  get a short int out of the database based on an identifier string

- const size_t & get_sizet (const String &entry_name) const
  get a size_t out of the database based on an identifier string

- const bool & get_bool (const String &entry_name) const
  get a bool out of the database based on an identifier string

- 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

- bool is_null () const
  function to check dbRep (does this envelope contain a letter)

Protected Member Functions

- ProblemDescDB (BaseConstructor, ParallelLibrary &parallel_lib)
  derived class constructors - Coplien, p. 139)

- virtual void derived_manage_inputs (const char *dakota_input_file)
  This version reads from the dakota input filename passed in.
Protected Attributes

- **DataStrategy** strategySpec
  
  to strategy_kwhandler() or *insert_node()*

- **List**< DataMethod > dataMethodList
  
  or *insert_node()*

- **List**< DataModel > dataModelList
  
  or *insert_node()*

- **List**< DataVariables > dataVariablesList
  
  variables_kwhandler() or *insert_node()*

- **List**< DataInterface > dataInterfaceList
  
  interface_kwhandler() or *insert_node()*

- **List**< DataResponses > dataResponsesList
  
  responses_kwhandler() or *insert_node()*

- **size_t** strategyCntr
  
  counter for strategy specifications used in check_input

Private Member Functions

- **ProblemDescDB** *get_db** (ParallelLibrary &parallel_lib)
  
  correct letter class

- **void** send_db_buffer ()

  and dataResponsesList. Used by *manage_inputs()*. 

- **void** receive_db_buffer ()

  and dataResponsesList. Used by *manage_inputs()*. 

Private Attributes

- **ParallelLibrary** & parallelLib

  reference to the parallel_lib object passed from main

- **List**< DataMethod >::iterator dataMethodIter

  iterator identifying the active list node in dataMethodList

- **List**< DataModel >::iterator dataModelIter
8.97 ProblemDescDB Class Reference

iterator identifying the active list node in dataModelList

- List<DataVariables>::iterator dataVariablesIter
  iterator identifying the active list node in dataVariablesList

- List<DataInterface>::iterator dataInterfaceIter
  iterator identifying the active list node in dataInterfaceList

- 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 dbLocked
  set_db_list_nodes invocation

- ProblemDescDB * dbRep
  pointer to the letter (initialized only for the envelope)

- int referenceCount
  number of objects sharing dbRep

8.97.1 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 (delimited by a newline), 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.
8.97.2 Constructor & Destructor Documentation

8.97.2.1 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.

8.97.2.2 ProblemDescDB (ParallelLibrary & parallel_lib)

standard constructor
This is the primary 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.

8.97.2.3 ProblemDescDB (const ProblemDescDB & db)

copy constructor
Copy constructor manages sharing of dbRep and incrementing of referenceCount.

8.97.2.4 ~ProblemDescDB ()

destructor
Destructor decrements referenceCount and only deletes dbRep when referenceCount reaches zero.

8.97.2.5 ProblemDescDB (BaseConstructor, ParallelLibrary & parallel_lib) [protected]
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).

8.97.3 Member Function Documentation

8.97.3.1 ProblemDescDB operator= (const ProblemDescDB & db)

assignment operator

8.97.3.2   void manage_inputs (CommandLineHandler & cmd_line_handler)

passed with the "-input" option on the DAKOTA command line.
Manage command line inputs using the CommandLineHandler class and parse the input file.

8.97.3.3   void manage_inputs (const char * dakota_input_file)

This version reads from the dakota input filename passed in.
Parse the input file.

8.97.3.4   ProblemDescDB * get_db (ParallelLibrary & parallel_lib)  [private]

correct letter class
Initializes dbRep to the appropriate derived type. The standard derived class constructors are invoked.

The documentation for this class was generated from the following files:

  - ProblemDescDB.H
  - ProblemDescDB.C
8.98 PStudyDACE Class Reference

design of experiments methods.

Inheritance diagram for PStudyDACE::

```
  Iterator
   |
   v
 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 the iterator; portion of run_iterator()`

- **const Variables & variables_results () const**
  
  `return a single final iterator solution (variables)`

- **const Response & response_results () const**
  
  `return a single final iterator solution (response)`

- **void response_results_active_set (const ActiveSet &set)**
  
  `set the requested data for the final iterator response results`

- **void print_results (ostream &s) const**
  
  `print the final iterator results`

- **virtual void extract_trends ()=0**
Redefines the run_iterator virtual function for the PStudy/DACE branch.

- void update_best (const RealVector &vars, const Response &response, const int eval_num)
  
  compares current evaluation to best evaluation and updates best

Protected Attributes

- **Variables bestVariables**
  
  best variables found during the study

- **Response bestResponses**
  
  best responses found during the study

- **Real bestObjFn**
  
  best objective function found during the study

- **Real bestConViol**
  
  precedence over objective function reduction.

- **size_t numObjFns**
  
  number of objective functions

- **size_t numLSqTerms**
  
  number of least squares terms

- **RealVector multiObjWts**
  
  vector of multiobjective weights

8.98.1 Detailed Description

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.

8.98.2 Member Function Documentation

8.98.2.1 void run () [inline, protected, virtual]

run the iterator; portion of run_iterator()
Iterator supports a construct/pre-run/run/post-run/destroy progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

8.98.2.2 void print_results (ostream & s) const [protected, virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in post_run().

Reimplemented from Iterator.

The documentation for this class was generated from the following files:

- DakotaPStudyDACE.H
- DakotaPStudyDACE.C
8.99 **PSUADEDesignCompExp Class Reference**

Wrapper class for the PSUade library.

Inheritance diagram for PSUADEDesignCompExp:

```
  Iterator
   |     |
   |     |
  Analyzer
   |     |
   |     |
PStudyDACE
   |     |
   |     |
PSUADEDesignCompExp
```

**Public Member Functions**

- **PSUADEDesignCompExp** *(Model &model)*
  
  *primary constructor for building a standard DACE iterator*

- **~PSUADEDesignCompExp** ()
  
  *destructor*

- **void extract_trends** ()
  
  *Redefines the run_iterator virtual function for the PStudy/DACE branch.*

- **void sampling_reset** (int min_samples, bool all_data_flag, bool stats_flag)
  
  *reset sampling iterator*

- **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** (const 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 IntArray & varPartitionsSpec
  number of partitions in each variable direction

- int numPartitions
  number of partitions to pass to PSUADE (levels = partitions + 1)

- bool allDataFlag
  Iterator::all_variables() and Iterator::all_responses().

- size_t numDACERuns
  counter for number of run() executions for this object

- bool varyPattern
  but are still repeatable

- int originalSeed
  (allows repeatable results)

- int randomSeed
  current seed for the random number generator

8.99.1 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.
8.99.2 Constructor & Destructor Documentation

8.99.2.1 PSUADEDesignCompExp (Model & model)

primary constructor for building a standard DACE iterator
This constructor is called for a standard iterator built with data from probDescDB.

8.99.3 Member Function Documentation

8.99.3.1 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.
The documentation for this class was generated from the following files:

- PSUADEDesignCompExp.H
- PSUADEDesignCompExp.C

[90x712]8.99 PSUADEDesignCompExp Class Reference
8.100 RecastBaseConstructor Struct Reference

Public Member Functions

- RecastBaseConstructor (int=0)
  
  C++ structs can have constructors.

8.100.1 Detailed Description

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:

- global_defs.h
8.101  RecastModel Class Reference

in order to recast the form of its inputs and/or outputs.

Inheritance diagram for RecastModel::

```
pictures/1/1 RegentsFullscreen.png
```

Public Member Functions

  
  *standard constructor*

- **RecastModel** (Model &sub_model, const size_t &num_recast_primary_fns, const size_t &num_recast_secondary_fns, const size_t &recast_secondary_offset)

  *alternate constructor*

- ~RecastModel ()

  *destructor*


  *completes initialization of the RecastModel after alternate construction*

- void submodel_supports_estimated_derivatives (bool ssed_flag)

  *override the submodel’s derivative estimation behavior*
Protected Member Functions

- void derived_compute_response (const ActiveSet &set)
  (forward to subModel.compute_response())

- void derived_asynch_compute_response (const ActiveSet &set)
  (forward to subModel.asynch_compute_response())

- const ResponseArray & derived_synchronize ()
  (forward to subModel.synchronize())

- const IntResponseMap & derived_synchronize_nowait ()
  (forward to subModel.synchronize_nowait())

- 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 surrogate_function_indices (const IntSet &surr_fn_indices)
  forward to subModel

- void surrogate_bypass (bool bypass_flag)
  models contained within this model

- void build_approximation ()
  builds the subModel approximation

- bool build_approximation (const Variables & vars, const Response & response)
  builds the subModel approximation

- void update_approximation (const Variables & vars, const Response & response, bool rebuild_flag)
  updates the subModel approximation

- void update_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array, bool rebuild_flag)
  updates the subModel approximation

- void append_approximation (const Variables & vars, const Response & response, bool rebuild_flag)
  appends the subModel approximation

- void append_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array, bool rebuild_flag)
appends the subModel approximation

- `Array< 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

- `void print_coefficients (ostream &s, size_t index) const`
  print a particular set of approximation coefficients within the subModel

- `const RealVector & approximation_variances (const RealVector &c_vars)`
  retrieve the approximation variances from the subModel

- `const List< SurrogateDataPoint > & approximation_data (size_t index)`
  retrieve the approximation data from the subModel

- `void component_parallel_mode (short mode)`
  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`
  evaluation (request forwarded to subModel)

- `void derived_init_communicators (const 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 (const int &max_iterator_concurrency, bool recurse_flag=true)`
  set active parallel configuration within subModel

- `void derived_free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)`
  to subModel

- `void serve ()`
  Completes when a termination message is received from stop_servers().
• void stop_servers ()
  
  when RecastModel iteration is complete.

• const String & interface_id () const
  
  return the subModel interface identifier

• int evaluation_id () const
  
  forwarded to subModel

• void set_evaluation_reference ()
  
  (request forwarded to subModel)

• void print_evaluation_summary (ostream & s, bool minimal_header=false, bool relative_count=true) const
  
  forwarded to subModel

Private Member Functions

• void asv_mapping (const ShortArray & recast_asv, ShortArray & sub_model_asv)
  
  into sub_model_asv for use with 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
  
  subModel variables)

• bool nonlinearVarsMapping
  
  Hessians are managed per function, not per variable.

• bool respMapping
  
  are supplied

• Sizet2DArray primaryRespMapIndices
  
  to RecastModel Response).

• Sizet2DArray secondaryRespMapIndices
8.101 RecastModel Class Reference

8.101.1 Detailed Description

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.

8.101.2 Constructor & Destructor Documentation
8.101.2.1 **RecastModel** (*Model* & *sub_model*, const size_t & *num_recast_primary_fns*, const size_t & *num_recast_secondary_fns*, const 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 `userDefinedConstrains`. The resulting model is sufficiently complete for passing to an *Iterator*.

8.101.3 **Member Function Documentation**

8.101.3.1 `void initialize (const Sizet2DArray & vars_map_indices, bool nonlinear_vars_mapping, void(*)(const Variables & recast_vars, Variables & sub_model_vars) variables_map, void(*)(const ShortArray & recast_asv, ShortArray & sub_model_asv) asv_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.

8.101.3.2 `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.

The documentation for this class was generated from the following files:

- RecastModel.H
- RecastModel.C


8.102 Response Class Reference

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 IntArray & active_set_derivative_vector () const
  return the active set derivative vector
void active_set_derivative_vector (const IntArray &asdv)
  set the active set derivative vector

const String & responses_id () const
  return the response identifier

const StringArray & function_labels () const
  return the response function identifier strings

void function_labels (const StringArray &labels)
  set the response function identifier strings

const Real & function_value (const size_t &i) const
  return a function value

const RealVector & function_values () const
  return all function values

void function_value (const Real &function_val, const size_t &i)
  set a function value

void function_values (const RealVector &function_vals)
  set all function values

const RealBaseVector & function_gradient (const size_t &i) const
  return a function gradient

const RealMatrix & function_gradients () const
  return all function gradients

void function_gradient (const RealBaseVector &function_grad, const size_t &i)
  set a function gradient

void function_gradients (const RealMatrix &function_grads)
  set all function gradients

const RealMatrix & function_hessian (const size_t &i) const
  return a function Hessian

const RealMatrixArray & function_hessians () const
  return all function Hessians

void function_hessian (const RealMatrix &function_hessian, const size_t &i)
  set a function Hessian

void function_hessians (const RealMatrixArray &function_hessians)
set all function Hessians

- **void read** (istream &s)
  
  read a response object from an istream

- **void write** (ostream &s) const
  
  write a response object to an ostream

- **void read_annotated** (istream &s)
  
  read a response object in annotated format from an istream

- **void write_annotated** (ostream &s) const
  
  write a response object in annotated format to an ostream

- **void read_tabular** (istream &s)
  
  read responseRep::functionValues in tabular format from an istream

- **void write_tabular** (ostream &s) const
  
  write responseRep::functionValues in tabular format to an 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 copy_results (const Response &response)

different derivative array sizing between the two response objects.

void copy_results (const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealMatrixArray &source_fn_hessians, const ActiveSet &source_set)

object. Care is taken to allow different derivative array sizing.

void reshape (const size_t &num_fns, const 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

8.102.1 Detailed Description

Response provides the handle class.

The Response class is a container class for an abstract set of functions (functionValues) and their first (function-Gradients) and second (functionHessians) derivatives. The functions may involve objective and constraint functions (optimization data set), least squares terms (parameter estimation data set), or generic response functions (uncertainty quantification data set). 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.
8.102.2 Constructor & Destructor Documentation

8.102.2.1 Response ()

default constructor

Need a populated problem description database to build a meaningful Response object, so set the response-Rep=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.H
- DakotaResponse.C
8.103 ResponseRep Class Reference

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 (istream &s)
  read a responseRep object from an istream

- void write (ostream &s) const
  write a responseRep object to an ostream

- void read.annotated (istream &s)
  read a responseRep object from an istream (annotated format)

- void write.annotated (ostream &s) const
  write a responseRep object to an ostream (annotated format)

- void read.tabular (istream &s)
  read functionValues from an istream (tabular format)

- void write.tabular (ostream &s) const
  write functionValues to an 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 ()
   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 copy_results (const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealMatrixArray &source_fn_hessians, const ActiveSet &source_set)
   update this response object from components of another response object

void reshape (const size_t &num_fns, const size_t &num_params, bool grad_ag, bool hess_ag)
   reshapes 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)
   of response functions

void active_set_derivative_vector (const IntArray &asdv)
   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*

- **RealMatrixArray 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 idResponses**

  *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*

---

### 8.103.1 Detailed Description

**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, function-Hessians, 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).

### 8.103.2 Constructor & Destructor Documentation

**8.103.2.1 ResponseRep (const Variables & vars, const ProblemDescDB & problem_db) [private]**

standard constructor built from problem description database

The standard constructor used by Dakota::ModelRep.
8.103.2.2 **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/recv buffers. However, NPSOLOptimizer’s user-defined functions option uses this constructor to build best-Responses and bestResponses needs functionLabels for I/O, so construction of functionLabels has been added.

8.103.3 **Member Function Documentation**

8.103.3.1 **void read** (istream & *s*)  [private]

read a responseRep object from an 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.).

8.103.3.2 **void write** (ostream & *s*) const  [private]

write a responseRep object to an ostream

ASCII version of write.

8.103.3.3 **void read.annotated** (istream & *s*)  [private]

read a responseRep object from an 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.

8.103.3.4 **void write.annotated** (ostream & *s*) const  [private]

write a responseRep object to an 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.

8.103.3.5 **void read.tabular** (istream & *s*)  [private]

read functionValues from an 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.
8.103.3.6 void write_tabular (ostream & s) const [private]

write functionValues to an ostream (tabular format)

write_tabular is used for output of functionValues in a tabular format for convenience in post-processing/plotting of DAKOTA results.

8.103.3.7 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.

8.103.3.8 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.

8.103.3.9 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.

8.103.3.10 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.

8.103.3.11 void copy_results (const RealVector & source_fn_vals, const RealMatrix & source_fn_grads, const RealMatrixArray & 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.
8.103.3.12  void reshape (const size_t & num_fns, const size_t & num_params, bool grad_flag, bool hess_flag) [private]

rephases response data arrays
Reshape functionValues, functionGradients, and functionHessians according to num_fns, num_params, grad_flag, and hess_flag.

8.103.3.13  void reset () [private]

resets all response data to zero
Reset all numerical response data (not labels, ids, or active set) to zero.

8.103.3.14  void reset_inactive () [private]

resets all inactive response data to zero
Used to clear out any inactive data left over from previous evaluations.

8.103.4  Member Data Documentation

8.103.4.1  RealMatrix functionGradients [private]

first derivatives of the response functions
the gradient vectors (plural) are arranged as a Jacobian matrix (singular) with (row, col) = (response fn index, variable index).
The documentation for this class was generated from the following files:

- DakotaResponse.H
- DakotaResponse.C
### 8.104 SingleMethodStrategy Class Reference

single model.

Inheritance diagram for SingleMethodStrategy:

```
        Strategy
            |       
            v       
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*
8.104.1 Detailed Description

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.H
- SingleMethodStrategy.C
8.105 SingleModel Class Reference

variables into responses.

Inheritance diagram for SingleModel:

![Inheritance Diagram]

Public Member Functions

- **SingleModel (ProblemDescDB &problem_db)**
  constructor

- **~SingleModel ()**
  destructor

Protected Member Functions

- **Interface & interface ()**
  return userDefinedInterface

- **void derived_compute_response (const ActiveSet &set)**
  (invokes a synchronous map() on userDefinedInterface)

- **void derived_asynch_compute_response (const ActiveSet &set)**
  (invokes an asynchronous map() on userDefinedInterface)

- **const ResponseArray & derived_synchronize ()**
  (invokes synch() on userDefinedInterface)

- **const IntResponseMap & derived_synchronize_nowait ()**
  (invokes synch_nowait() on userDefinedInterface)

- **void component_parallel_mode (short mode)**
  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
  evaluation (request forwarded to userDefinedInterface)

- void derived_init_communicators(const int &max_iterator_concurrency, bool recurse_flag=true)
  userDefinedInterface)

- void derived_init_serial()
  userDefinedInterface).

- void derived_set_communicators(const int &max_iterator_concurrency, bool recurse_flag=true)
  (request forwarded to userDefinedInterface)

- void derived_free_communicators(const int &max_iterator_concurrency, bool recurse_flag=true)
  (request forwarded to userDefinedInterface)

- void serve()
  Completes when a termination message is received from stop_servers().

- void stop_servers()
  operations when SingleModel iteration is complete.

- const String & interface_id() const
  return the userDefinedInterface identifier

- int evaluation_id() const
  (request forwarded to userDefinedInterface)

- void set_evaluation_reference()
  (request forwarded to userDefinedInterface)

- void print_evaluation_summary(ostream &, bool minimal_header=false, bool relative_count=true) const
  (request forwarded to userDefinedInterface)

**Private Attributes**

- Interface userDefinedInterface
  the interface used for mapping variables to responses
8.105.1 Detailed Description

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.

The documentation for this class was generated from the following files:

- SingleModel.H
- SingleModel.C
8.106  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 (const RealVector &local_fn_vals, NEWMAT::ColumnVector &g, const size_t &offset)**
  - constraint evaluator functions

- **void copy_con_vals (const NEWMAT::ColumnVector &g, RealVector &local_fn_vals, const size_t &offset)**
  - final solution logging

- **void copy_con_grad (const RealMatrix &local_fn_grads, NEWMAT::Matrix &grad_g, const size_t &offset)**
  - used by constraint evaluator functions

- **void copy_con_hess (const RealMatrixArray &local_fn_hessians, OPTPP::OpptArray<NEWMAT::SymmetricMatrix > &hess_g, const size_t &offset)**
  - used by constraint evaluator functions

- **void snll_pre_instantiate (const String &merit_fn, bool bound_constr_flag, const int &num_constr)**
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, const int &num_constr, bool debug_output, OPTPP::OptimizeClass *the_optimizer, OPTPP::NLP0 *nlf_objective, OPTPP::FDNLF1 *fd_nlf1, OPTPP::FDNLF1 *fd_nlf1_con)

method instantiation


method invocation

- void **snll_post_run** (OPTPP::NLP0 *nlf_objective)

method instantiations

Static Protected Member Functions

- static void **init_fn** (int n, NEWMAT::ColumnVector &x)

  An initialization mechanism provided by OPT++ (not currently used).

Protected Attributes

- **String** searchMethod

  trust_region, or tr_pds

- OPTPP::SearchStrategy **searchStrat**

  enum: LineSearch, TrustRegion, or TrustPDS

- OPTPP::MeritFcn **meritFn**

  enum: NormFmu, ArgaezTapia, or VanShanno

- bool **constantASVFlag**

  this into mode override, reliance on duplicate detection can be avoided.

Static Protected Attributes

- static **Minimizer** *optLSqInstance*

  evaluator functions in order to avoid the need for static data
8.106 SNLLBase Class Reference

- static bool modeOverrideFlag
  
  *Hessian requests.*

- static EvalType lastFnEvalLocn
  
  *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.*

### 8.106.1 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.H
- SNLLBase.C
8.107 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)
  *constructor*

- **~SNLLLeastSq** ()
  *destructor*

- **void minimize_residuals** ()
  *Performs the iterations to determine the least squares solution.*

Protected Member Functions

- **virtual void derived_pre_run** ()
  *invokes SNLLBase::snll_pre_run() and performs other set-up*

- **virtual void derived_post_run** ()
  *invokes SNLLBase::snll_post_run() and performs other solution processing*

Static Private Member Functions

- **static void nlf2_evaluator_gn** (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, NEWMAT::SymmetricMatrix &hess_f, int &result_mode)
value, gradient, and Hessian using the Gauss-Newton approximation.

- static void `constraint1_evaluator_gn` (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, int &result_mode)
  values and gradients to OPT++ Gauss-Newton methods.

- static void `constraint2_evaluator_gn` (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, OPTPP::OptppArray<NEWMAT::SymmetricMatrix> &hess_g, int &result_mode)
  values, gradients, and Hessians to OPT++ Gauss-Newton methods.

### Private Attributes

- **SNLLLeastSq * prevInstance**
  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.
Static Private Attributes

- static SNLLLeastSq * snllLsqInstance

functions in order to avoid the need for static data

8.107.1 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/methods/OPTPP directory for information on OPT++ class member functions.

8.107.2 Member Function Documentation

8.107.2.1 void nlf2_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, NEWMAT::ColumnVector & grad_f, NEWMAT::SymmetricMatrix & hess_f, int & result_mode) [static, private]

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).
8.107.2.2 `void constraint1_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector & x, 
NEWMAT::ColumnVector & g, NEWMAT::Matrix & grad_g, int & result_mode)` [static, private]

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.

8.107.2.3 `static void constraint2_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector & x, 
NEWMAT::ColumnVector & g, NEWMAT::Matrix & grad_g, OPTPP::OptppArray<
NEWMAT::SymmetricMatrix > & hess_g, int & result_mode)` [static, private]

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.

The documentation for this class was generated from the following files:

- SNLLLeastSq.H
- SNLLLeastSq.C
8.108  SNLLOptimizer Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLOptimizer::

![Inheritance Diagram](image)

Public Member Functions

- **SNLLOptimizer (Model &model)**
  
  *standard constructor*

- **SNLLOptimizer (Model &model, const String &method)**
  
  *alternate constructor for instantiations "on the fly”*

- **SNLLOptimizer (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 mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode), void(*user_con_eval)(int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &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

- virtual void derived_pre_run ()
  
  invokes SNLLBase::snll_pre_run() and performs other set-up

Static Private Member Functions

- static void nlf0_evaluator (int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, int &result_mode)
  
  require only function values.

- static void nlf1_evaluator (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode)
  
  values and gradients to OPT++ methods.

- static void nlf2_evaluator (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, NEWMAT::SymmetricMatrix &hess_f, int &result_mode)
  
  values, gradients, and Hessians to OPT++ methods.

- static void constraint0_evaluator (int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, int &result_mode)
  
  only constraint values.

- static void constraint1_evaluator (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, int &result_mode)
  
  values and gradients to OPT++ methods.

- static void constraint2_evaluator (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, OPTPP::OptppArray< NEWMAT::SymmetricMatrix > &hess_g, int &result_mode)
  
  values, gradients, and Hessians to OPT++ methods.

Private Attributes

- SNLLOptimizer * prevInstance
  
  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
  NonDExtract 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
  functions in order to avoid the need for static data

### 8.108.1 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/methods/OPTPP directory for information on OPT++ class member functions.

8.108.2 Constructor & Destructor Documentation

8.108.2.1 SNLLOptimizer (Model & model)

standard constructor

This constructor is used for normal instantiations using data from the ProblemDescDB.

8.108.2.2 SNLLOptimizer (Model & model, const String & method)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

8.108.2.3 SNLLOptimizer (const RealVector & initial_point, const RealVector & var_lower_bnds, const RealVector & var_upper_bnds, const RealMatrix & lin_ineq_coefs, const RealVector & lin_ineq_lower_bnds, const RealVector & lin_ineq_upper_bnds, const RealMatrix & lin_eq_coefs, const RealVector & lin_eq_targets, const RealVector & nonlin_ineq_lower_bnds, const RealVector & nonlin_ineq_upper_bnds, const RealVector & nonlin_eq_targets, void(*)(int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode) user_obj_eval, void(*)(int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Matrix &grad_g, int &result_mode) user_con_eval)

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.

8.108.3 Member Function Documentation
void nlf0_evaluator (int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, int & result_mode) [static, private]

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).

void nlf1_evaluator (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, NEWMAT::ColumnVector & grad_f, int & result_mode) [static, private]

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.

void nlf2_evaluator (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, NEWMAT::ColumnVector & grad_f, NEWMAT::SymmetricMatrix & hess_f, int & result_mode) [static, private]

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 nlf1. Gauss-Newton does not fit this model; it uses nlf2_evaluator_gn instead of nlf2_evaluator.

void constraint0_evaluator (int n, const NEWMAT::ColumnVector & x, NEWMAT::ColumnVector & g, int & result_mode) [static, private]

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).

void constraint1_evaluator (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::ColumnVector & g, NEWMAT::Matrix & grad_g, int & result_mode) [static, private]

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.

The documentation for this class was generated from the following files:

- SNLLOptimizer.H
• SNLLOptimizer.C
8.109 SOLBase Class Reference

Base class for Stanford SOL software.

Inheritance diagram for SOLBase::

```
    SOLBase
    |   |
    v   v
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 int &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_ag, bool vendor_num_grad_ag, bool verbose_output, 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 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 *cjac, int &nstate)

  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
  nonlinear constraint bounds)

- double * linConstraintMatrixF77
  \([A]\) matrix from NPSOL manual: linear constraint coefficients

- double * upperFactorHessianF77
  the Lagrangian.

- double * constraintJacMatrixF77
  \([CJAC]\) matrix from NPSOL manual: nonlinear constraint Jacobian

- int fnEvalCntr
  counter for testing against maxFunctionEvals

- size_t constrOffset
  and \texttt{NPSOL::Optimizer::numObjectiveFunctions}

### Static Protected Attributes

- static SOLBase * solInstance
  functions in order to avoid the need for static data

- static Minimizer * optLSqInstance
  evaluator functions in order to avoid the need for static data

### 8.109.1 Detailed Description

Base class for Stanford SOL software.

The \texttt{SOLBase} class provides a common base class for \texttt{NPSOL::Optimizer} and \texttt{NLSSOL::LeastSq}, 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:

- \texttt{SOLBase.H}
- \texttt{SOLBase.C}
8.110 Strategy Class Reference

Base class for the strategy class hierarchy.

Inheritance diagram for Strategy::

```
Strategy

ConcurrentStrategy MultilevelOptStrategy SingleMethodStrategy SurrBasedOptStrategy
```

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 model(s). Called from main.C.*

- virtual const Variables & variables_results () const
  
  *return the final strategy solution (variables)*

- virtual const Response & response_results () const
  
  *return the final strategy solution (response)*

- void run_iterator (Iterator &the_iterator, Model &the_model)
  
  *due to use by MINLPNode.*

- **ProblemDescDB & problem_description_db () const**
  
  *returns the problem description database (probDescDB)*
Protected Member Functions

- **Strategy** (BaseConstructor, ProblemDescDB &problem_db)
  derived class constructors - Coplien, p. 139

- void **init_communicators** (Iterator &the_iterator, Model &the_model)
  convenience function for allocating comms prior to running an iterator

- void **free_communicators** (Iterator &the_iterator, Model &the_model)
  convenience function for deallocating comms after running an iterator

- void **initialize_graphics** (const Model &model)
  data tabulation

Protected Attributes

- ProblemDescDB & probDescDB
  class member reference to the problem description database

- ParallelLibrary & parallelLib
  class member reference to the parallel library

- String **strategyName**
  opt_under_uncertainty, branch_and_bound, multi_start, or pareto_set.

- 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

- bool **mpirunFlag**
  flag for parallel MPI launch of DAKOTA

- bool **graphicsFlag**
  flag for using graphics in a graphics executable

- bool **tabularDataFlag**
  flag for file tabulation of graphics data
- **String tabularDataFile**
  
  *filename for tabulation of graphics data*

**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*

**8.110.1 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.

**8.110.2 Constructor & Destructor Documentation**

**8.110.2.1 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.
8.110.2.2  **Strategy (ProblemDescDB & problem_db)**

envelope constructor
Used in `main.C` 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.

8.110.2.3  **Strategy (const Strategy & strat)**

copy constructor
Copy constructor manages sharing of strategyRep and incrementing of referenceCount.

8.110.2.4  **~Strategy () [virtual]**

destructor
Destructor decrements referenceCount and only deletes strategyRep when referenceCount reaches zero.

8.110.2.5  **Strategy (BaseConstructor, ProblemDescDB & problem_db) [protected]**

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`).

8.110.3  **Member Function Documentation**

8.110.3.1  **Strategy operator= (const Strategy & strat)**

assignment operator

8.110.3.2  **void run_iterator (Iterator & the_iterator, Model & the_model)**

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.

8.110.3.3 void init_communicators (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.

8.110.3.4 void free_communicators (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.

8.110.3.5 void initialize_graphics (const Model & model) [protected]

data tabulation
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.

8.110.3.6 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.
The documentation for this class was generated from the following files:

- DakotaStrategy.H
- DakotaStrategy.C
8.111 String Class Reference

Dakota::String class, used as main string class for Dakota.

Public Member Functions

- **String ()**
  Default constructor.

- **String (const String &a)**
  Copy constructor for incoming String.

- **String (const String &a, size_t start_index, size_t num_items)**
  Copy constructor for portion of incoming String.

- **String (const char *c_string)**
  Copy constructor for incoming char* array.

- **String (const DAKOTA_BASE_STRING &a)**
  Copy constructor for incoming base string.

- **~String ()**
  Destructor.

- **String & operator= (const String &)**
  Assignment operator for incoming String.

- **String & operator= (const DAKOTA_BASE_STRING &)**
  Assignment operator for incoming base string.

- **String & operator= (const char *)**
  Assignment operator for incoming char* array.

- **operator const char * () const**
  The operator() returns pointer to standard C char array.

- **String & toUpper ()**
  Convert to upper case string.

- **void upper ()**

- **String & toLower ()**
  Convert to lower case string.
void lower ()

bool contains (const char *sub_string) const

    Returns true if String contains char* substring.

bool begins (const char *sub_string) const

    Returns true if String starts with char* substring.

bool ends (const char *sub_string) const

    Returns true if String ends with char* substring.

char * data () const

    Returns pointer to standard C char array.

8.111.1 Detailed Description

Dakota::String class, used as main string class for Dakota.

The Dakota::String class is the common string class for Dakota. It provides a common interface for string operations whether inheriting from the STL basic_string or the Rogue Wave RWCString class

8.111.2 Member Function Documentation

8.111.2.1 operator const char * () const [inline]

The operator() returns pointer to standard C char array.

The operator () returns a pointer to a char string. Uses the STL c_str() method. This allows for the String to be used in method calls without having to call the data() or c_str() methods.

8.111.2.2 void upper ()

Private method which converts String to upper. Utilizes an STL iterator to step through the string and then calls the STL toupper() method. Needs to be done this way because STL only provides a single char toupper method.

8.111.2.3 void lower ()

Private method which converts String to lower. Utilizes an STL iterator to step through the string and then calls the STL tolower() method. Needs to be done this way because STL only provides a single char tolower method.
8.111.2.4 bool contains (const char * sub_string) const  [inline]

Returns true if String contains char* substring.
Returns true if the String contains the char* sub_string. Uses the STL find() method.

8.111.2.5 bool begins (const char * sub_string) const  [inline]

Returns true if String starts with char* substring.
Returns true if the String begins with the char* sub_string. Uses the STL compare() method.

8.111.2.6 bool ends (const char * sub_string) const  [inline]

Returns true if String ends with char* substring.
Returns true if the String ends with the char* sub_string. Uses the STL compare() method.

8.111.2.7 char * data () const  [inline]

Returns pointer to standard C char array.
Returns a pointer to C style char array. Needed to mimic the Rogue Wave string class. USE WITH CARE.
The documentation for this class was generated from the following files:

- DakotaString.H
- DakotaString.C
8.112 SurfpackApproximation Class Reference

Interface between Surfpack and Dakota.

Inheritance diagram for SurfpackApproximation::

```
Approximation

SurfpackApproximation
```

Public Member Functions

- **SurfpackApproximation ()**
  
  *default constructor*

- **SurfpackApproximation (ProblemDescDB &problem_db, const size_t &num_acv)**
  
  *the Dakota/Surfpack boundary*

- **~SurfpackApproximation ()**
  
  *destructor*

Protected Member Functions

- **int num_coefficients () const**
  
  *derived class approximation type in numVars dimensions*

- **void find_coefficients ()**
  
  *and the appropriate Surfpack build method will be invoked*

- **const Real & get_value (const RealVector &x)**
  
  *Return the value of the Surfpack surface for a given parameter vector x.*

- **const RealBaseVector & get_gradient (const RealVector &x)**
  
  *retrieve the approximate function gradient for a given parameter vector x*

- **const RealMatrix & get_hessian (const RealVector &x)**
  
  *retrieve the approximate function Hessian for a given parameter vector x*
Private Member Functions

- void checkForEqualityConstraints ()
  
  *point, gradient, and/or hessian*

- SurfData * surrogates_to_surf_data ()
  
  *copy from SurrogateDataPoint to SurfPoint/SurfData*

Private Attributes

- Surface * surface
  
  *The native Surfpack approximation.*

- SurfData * surfData
  
  *The data used to build the approximation, in Surfpack format.*

8.112.1 Detailed Description

*Interface between Surfpack and Dakota.*

The SurfpackApproximation class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SurfpackApproximation builds a Surfpack Surface object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial neural networks, radial basis function network, or multivariate adaptive regression splines (MARS).

8.112.2 Constructor & Destructor Documentation

8.112.2.1 SurfpackApproximation (ProblemDescDB & problem_db, const size_t & num_acv)

initialize the Dakota/Surfpack boundary

Initialize the embedded Surfpack surface object and configure it using the specifications from the input file. Data for the surface is created later.

Todo

Add RBFNet surface fit interface

8.112.3 Member Function Documentation
8.112.3.1 void find_coefficients () [protected, virtual]

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.

8.112.3.2 const RealMatrix & get_hessian (const RealVector & x) [protected, virtual]

retrieve the approximate function Hessian for a given parameter vector x.

Todo
Make this acceptably efficient

Reimplemented from Approximation.

8.112.3.3 void checkForEqualityConstraints () [private]

point, gradient, and/or hessian
If there is an anchor point, add an equality constraint for its response value. Also add constraints for gradient and hessian, if applicable.

Todo
improve efficiency of conversion.

8.112.3.4 SurfData * surrogates_to_surf_data () [private]

copy from SurrogateDataPoint to SurfPoint/SurfData
Copy the data stored in Dakota-style SurrogateDataPoint objects into Surfpack-style SurfPoint and SurfData objects.
The documentation for this class was generated from the following files:
• SurfpackApproximation.H
• SurfpackApproximation.C
8.113 **SurrBasedOptStrategy Class Reference**

*Strategy* for provably-convergent surrogate-based optimization.

Inheritance diagram for SurrBasedOptStrategy::

```
Strategy
  ↓
SurrBasedOptStrategy
```

**Public Member Functions**

- **SurrBasedOptStrategy** (ProblemDescDB &problem_db)
  *constructor*
- **~SurrBasedOptStrategy** ()
  *destructor*
- **void run_strategy** ()
  *global, or hierarchical surrogates over a series of trust regions.*
- **const Variables & variables_results** () const
  *return the SBO final solution (variables)*
- **const Response & response_results** () const
  *return the SBO final solution (response)*

**Private Member Functions**

- **void run_surrogate_based_optimization** ()
  *the core SBO algorithm, as called from run_strategy()*
- **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)
    merit function near zero

void tr_ratio_check(const RealVector &c_vars_star, const RealVector &tr_lower_bounds, const RealVector &tr_upper_bounds)
    next trust region 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 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 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 RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealBaseVector &lag_grad)
    compute the gradient of the Lagrangian function

Real augmented_lagrangian_merit(const RealVector &fn_vals, 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 RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealBaseVector &alag_grad)
    compute the gradient of the augmented Lagrangian function

Real penalty_merit(const RealVector &fn_vals)
    compute a penalty function from a set of function values

void penalty_gradient(const RealVector &fn_vals, const RealMatrix &fn_grads, RealBaseVector &pen_grad)
    compute the gradient of the penalty function
• **Real** objective (const **RealVector** &fn_vals)
  
  *compute a composite objective value from one or more objective functions*

• **void** objective_gradient (const **RealMatrix** &fn_grads, **RealBaseVector** &obj_grad)

  *compute the gradient of the composite objective function*

• **Real** constraintViolation (const **RealVector** &fn_vals, const **Real** &constraint_tol)

  *compute the constraint violation from a set of function values*

• **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 **Response** &surrogate_response, **Response** &recast_response)

  *static function used to define the approximate subproblem constraints.*

• **static void** hom_objective_eval (int &mode, int &n, double *tau_and_x, double &f, double *grad_f, int &)

  *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)

  *homotopy constraint relaxation formulation.*

### Private Attributes

• **Model** surrogateModel

  *the surrogate model (a SurrogateModel object)*

• **Iterator** selectedIterator

  *the optimizer used on surrogateModel*

• **Real** trustRegionFactor

  *bound - lower bound for each design variable*.

• **Real** minTrustRegionFactor
8.113 SurrBasedOptStrategy Class Reference

factor is reduced below the value of minTrustRegionFactor

- Real convergenceTol
  
soft convergence checks

- Real constraintTol
  
be a violated constraint.

- 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

- Real gammaNoChange
  
factor for maintaining the current trust region size (normally 1.0)

- short approxSubProbObj
  
or AUGMENTED_LAGRANGIAN_OBJ

- short approxSubProbCon
  
ORIGINAL_CON.

- Model approxSubProbModel
  
envelope object used to keep RecastModel instance in scope

- short trConstraintRelax
  
points: NO_RELAX or HOMOTOPY

- short meritFnType
  
ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, or AUGMENTED_LAGRANGIAN_MERIT.

- short acceptLogic
  
type of iterate acceptance test logic: FILTER or TR_RATIO

- RealVectorList sboFilter
  
constraint violation) for iterate selection/rejection

- RealVector lagrangeMult
  
Lagrange multipliers for basic Lagrangian calculations.
- **RealVector** augLagrangeMult
  
  *Lagrange multipliers for augmented Lagrangian calculations.*

- **Real** penaltyParameter
  
  *penalty calculations; increased in update_penalty()*

- **int** penaltyIterOffset
  
  *for adaptive_penalty merit functions*

- **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
  
  *Lagrangian updates (refer to Conn, Gould, and Toint, section 14.4).*

- **int** sboIterNum
  
  *SBO iteration number.*

- **int** sboIterMax
  
  *maximum number of SBO iterations*

- **short** convergenceFlag
  
  *code indicating satisfaction of hard or soft convergence conditions*

- **size_t** numFns
  
  *number of response functions*

- **size_t** numVars
  
  *number of active continuous variables*

- **short** softConvCount
  
  *count reaches softConvLimit, stop SBO.*

- **short** softConvLimit
  
  *exceeded by softConvCount, stop SBO.*

- **bool** truthGradientFlag
  
  *flags the use/availability of truth gradients within the SBO process*
- `bool approxGradientFlag`  
  flags the use/availability of surrogate gradients within the SBO process

- `bool truthHessianFlag`  
  flags the use/availability of truth Hessians within the SBO process

- `bool approxHessianFlag`  
  flags the use/availability of surrogate Hessians within the SBO process

- `bool correctionFlag`  
  flags the use of a multipoint data fit surrogate (TANA)

- `bool globalApproxFlag`  
  flags the use of a global data fit surrogate (rsm, ann, mars, kriging)

- `bool multiptApproxFlag`  
  flags the use of a local data fit surrogate (Taylor series)

- `bool hierarchApproxFlag`  
  flags the use of a model hierarchy/multifidelity surrogate

- `bool newCenterFlag`  
  a new trust region center

- `bool daceCenterPtFlag`  
  evaluations for global approximations (CCD, Box-Behnken)

- `bool multiLayerBypassFlag`  
  (responseCenterTruth and responseStarTruth).

- `bool useGradsFlag`  
  to be evaluated for each DACE point in global surrogate builds.

- `size_t numObjFns`  
  number of objective functions

- `size_t numNonlinIneqConstr`  
  number of nonlinear inequality constraints

- `size_t numNonlinEqConstr`  
  number of nonlinear equality constraints

- `RealVector multiObjWts`
vector of multiobjective weights.

- **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 bigRealBoundSize**
  cutoff value for continuous bounds

- **RealVector nonlinIneqLowerBndsSlack**
  individual violations of nonlinear inequality constraint lower bounds

- **RealVector nonlinIneqUpperBndsSlack**
  individual violations of nonlinear inequality constraint upper bounds

- **RealVector nonlinEqTargetsSlack**
  individual violations of nonlinear equality constraint targets

- **Real tau**
  constraint relaxation parameter

- **Real alpha**
  constraint relaxation parameter backoff parameter (multiplier)

- **Variables varsCenter**
  variables at the trust region center

- **Response responseCenterApprox**
  approx response at trust region center

- **Response responseStarApprox**
  approx response at SBO cycle optimum

- **Response responseCenterTruth**
  truth response at trust region center

- **Response responseStarTruth**
  truth response at SBO cycle optimum

- **Variables bestVariables**
  best variables found in SBO
8.113 SurrBasedOptStrategy Class Reference

- **Response bestResponses**
  
  *best responses found in SBO*

**Static Private Attributes**

- static SurrBasedOptStrategy * sboInstance
  
  *pointer to SBO strategy used in static member functions*

### 8.113.1 Detailed Description

**Strategy** for provably-convergent surrogate-based optimization.

This strategy uses a SurrogateModel to perform optimization 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.

### 8.113.2 Member Function Documentation

#### 8.113.2.1 void run_strategy () [virtual]

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 optimizer 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.

Reimplemented from Strategy.

#### 8.113.2.2 void hard_convergence_check (const Response & response_truth, const RealVector & e_vars, const RealVector & lower_bnds, const RealVector & upper_bnds) [private]

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.

#### 8.113.2.3 void tr_ratio_check (const RealVector & c_vars_star, const RealVector & tr_lower_bounds, const RealVector & tr_upper_bounds) [private]

next trust region and check for soft convergence (diminishing returns)
Assess acceptance of SBO 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).

8.113.2.4 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 SBO 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.

8.113.2.5 void update_lagrange_multipliers (const RealVector & fn_vals, const RealMatrix & fn_grads) [private]

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.

8.113.2.6 void update_augmented_lagrange_multipliers (const RealVector & fn_vals) [private]

initialize and update the Lagrange multipliers for augmented Lagrangian

For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through solution of a nonnegative linear least squares problem.

8.113.2.7 bool update_filter (const RealVector & fn_vals) [private]

update a filter from a set of function values

Update the sboFilter with fn_vals if new iterate is non-dominated.

8.113.2.8 Real lagrangian_merit (const RealVector & fn_vals, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts) [private]

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<=0 and h=0. The bounds/targets passed in may reflect the original constraints or the relaxed constraints.
8.113.2.9  Real augmented_Lagrangian_merit (const RealVector & fn_vals, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts) [private]

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 \( g^e \leq 0 \) and \( h=0 \). The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

8.113.2.10  Real penalty_merit (const RealVector & fn_vals) [private]

compute a penalty function from a set of function values

The penalty function computation applies a quadratic penalty to any constraint violations and adds this to the objective function(s) \( p = f + r_p \cdot cv \).

8.113.2.11  Real objective (const RealVector & fn_vals) [private]

compute a composite objective value from one or more objective functions

The composite objective computation sums up the contributions from one of more objective functions using the multiobjective weights.

8.113.2.12  void objective_gradient (const RealMatrix & fn_grads, RealBaseVector & obj_grad) [private]

compute the gradient of the composite objective function

The composite objective gradient computation sums up the contributions from one of more objective function gradients using the multiobjective weights.

8.113.2.13  Real constraint_violation (const RealVector & fn_vals, const Real & constraint_tol) [private]

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).

8.113.2.14  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.
8.113.2.15  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.

8.113.2.16  void hom_objective_eval (int & mode, int & n, double * tau_and_x, double & f, double * grad_f, int &) [static, private]

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.

8.113.2.17  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, private]

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.

The documentation for this class was generated from the following files:

- SurrBasedOptStrategy.H
- SurrBasedOptStrategy.C
8.114 SurrogateDataPoint Class Reference

for defining a "truth" data point.

Public Member Functions

- **SurrogateDataPoint ()**
  *default constructor*

- **SurrogateDataPoint (const RealVector &x, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)**
  *standard constructor*

- **SurrogateDataPoint (const SurrogateDataPoint &sdp)**
  *copy constructor*

- ~**SurrogateDataPoint ()**
  *destructor*

- **SurrogateDataPoint & operator= (const SurrogateDataPoint &sdp)**
  *assignment operator*

- bool **operator== (const SurrogateDataPoint &sdp) const**
  *equality operator*

- const **RealVector & continuous_variables () const**
  *return continuousVars*

- const Real & **response_function () const**
  *return responseFn*

- const **RealBaseVector & response_gradient () const**
  *return responseGrad*

- const **RealMatrix & response_hessian () const**
  *return responseHess*

- bool **is_null () const**
  *function to check sdpRep (does this handle contain a body)*
Private Attributes

- **SurrogateDataPointRep * sdpRep**
  
  *pointer to the body (handle-body idiom)*

8.114.1 Detailed Description

for defining a “truth” data point.

A list of these data points is contained in each \texttt{Approximation} instance (\texttt{Approximation::currentPoints}) and provides the data to build the approximation. A handle-body idiom is used to avoid excessive data copying overhead.

The documentation for this class was generated from the following file:

- DakotaApproximation.H
8.115  SurrogateDataPointRep Class Reference

or body, may be shared by multiple SurrogateDataPoint handle instances.

Private Member Functions

- SurrogateDataPointRep (const RealVector &x, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)
  
  constructor

- ~SurrogateDataPointRep ()
  
  destructor

Private Attributes

- RealVector continuousVars
  
  continuous variables

- Real responseFn
  
  truth response function value

- RealBaseVector responseGrad
  
  truth response function gradient

- RealMatrix responseHess
  
  truth response function Hessian

- int referenceCount
  
  number of handle objects sharing sdpRep

Friends

- class SurrogateDataPoint
  
  the handle class can access attributes of the body class directly
8.115.1 Detailed Description

or body, may be shared by multiple SurrogateDataPoint handle instances.
The SurrogateDataPoint/SurrogateDataPointRep pairs utilize a handle-body idiom (Coplien, Advanced C++).
The documentation for this class was generated from the following file:

- DakotaApproximation.H
8.116 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** (const pair< short, short > &view, const ActiveSet &set, const String &corr_type, const short &corr_order)
  
  Alternate constructor

- ~SurrogateModel ()
  
  Destructor

- void **compute_correction** (const Response &truth_response, const Response &approx_response, const RealVector &c_vars)
  
  Agreement with truth_response

- void **apply_correction** (Response &approx_response, const RealVector &c_vars, bool quiet_flag=false)
  
  Apply the correction computed in compute_correction() to approx_response

- void **auto_correction** (bool correction_flag)
  
  Sets autoCorrection to on (true) or off (false)

- bool **auto_correction** ()
  
  Returns autoCorrection setting

- void **check_submodel_compatibility** (const Model &sub_model)
  
  HierarchSurrModel::highFidelityModel.

- bool **force_rebuild** ()
forced based on changes in the inactive data

- void `asv_mapping` (const `ShortArray &orig_asv`, `ShortArray &actual_asv`, `ShortArray &approx_asv`, bool `build_flag`)  
  distributes the incoming `orig_asv` among `actual_asv` and `approx_asv`

- void `asv_mapping` (const `ShortArray &actual_asv`, const `ShortArray &approx_asv`, `ShortArray &combined_asv`)  
  reconstitutes a combined `asv` from `actual_asv` and `approx_asv`

- void `response_mapping` (const `Response &actual_response`, const `Response &approx_response`, `Response &combined_response`)  
  overlays `actual_response` and `approx_response` to update `combined_response`

- void `cached_mapping` (const `ResponseArray &orig_resp_array`, `IntResponseMap &cached_map`, const `Int-IntMap &id_map`, `ResponseArray &merged_array`)  
  inserts a cached response map into a response array in order

**Protected Attributes**

- bool `mixedResponseSet`  
  flag for mixed approximate/actual responses

- `IntSet surrogateFnIndices`  
  subset that is approximated

- `ResponseArray surrResponseArray`  
  array of surrogate responses used in `derived_synchronize()` functions

- `IntResponseMap surrResponseMap`  
  map of surrogate responses used in `derived_synchronize_nowait()` functions

- `IntRealVectorMap rawCVarsMap`  
  not contain lower level variables sets from finite differencing.

- `IntIntMap truthIdMap`  
  `DataFitSurrModel/HierarchSurrModel id.`

- `IntIntMap surrIdMap`  
  `DataFitSurrModel/HierarchSurrModel ids.`

- `IntResponseMap cachedApproxRespMap`  
  portions were still pending.

- `String correctionType`
approximation correction approach to be used: additive or multiplicative

- short correctionOrder
  approximation correction order to be used: 0, 1, or 2

- bool autoCorrection
  and HierarchSurrModel approximate response computations

- bool correctionComputed
  and is available for application

- size_t approxBuilds
  number of calls to build_approximation()

- bool surrogateBypass
  on the underlying truth model.

- RealVector fitCLBnds
  the approximation is built; used to detect when a rebuild is required.

- RealVector fitCUBnds
  the approximation is built; used to detect when a rebuild is required.

- IntVector fitDLBnds
  the approximation is built; used to detect when a rebuild is required.

- IntVector fitDUBnds
  the approximation is built; used to detect when a rebuild is required.

- RealVector fitInactCVars
  rebuild is required.

- IntVector fitInactDVars
  rebuild is required.

Private Member Functions

- void apply_additive_correction (RealVector &alpha_corrected_fns, RealMatrix &alpha_corrected_grads, RealMatrixArray &alpha_corrected_hessians, const RealVector &c_vars, const ActiveSet &set)
  internal convenience function for applying additive corrections

- void apply_multiplicative_correction (RealVector &beta_corrected_fns, RealMatrix &beta_corrected_grads, RealMatrixArray &beta_corrected_hessians, const RealVector &c_vars, const ActiveSet &set)
  internal convenience function for applying multiplicative corrections
Private Attributes

- `bool badScalingFlag`
  
  *corrections; triggers an automatic switch to additive corrections*

- `bool combinedFlag`
  
  *flag indicating the combination of additive/multiplicative corrections*

- `bool computeAdditive`
  
  *flag indicating the need for additive correction calculations*

- `bool computeMultiplicative`
  
  *flag indicating the need for multiplicative correction calculations*

- `RealVector addCorrFns`
  
  *high and low fidelity model values at x=x_center.*

- `RealMatrix addCorrGrads`
  
  *high/low function difference at x=x_center.*

- `RealMatrixArray addCorrHessians`
  
  *high/low function difference at x=x_center.*

- `RealVector multCorrFns`
  
  *high fidelity to low fidelity model values at x=x_center.*

- `RealMatrix multCorrGrads`
  
  *of the high/low function ratio at x=x_center.*

- `RealMatrixArray multCorrHessians`
  
  *of the high/low function ratio at x=x_center.*

- `RealVector combineFactors`
  
  *correction instead of a strictly local correction.*

- `RealVector correctionCenterPt`
  
  *(x - x_c) terms in 1st-/2nd-order corrections.*

- `RealVector correctionPrevCenterPt`
  
  *copy of correctionCenterPt from the previous correction cycle*

- `RealVector approxFnsCenter`
  
  *unavailable when applying 1st-/2nd-order multiplicative corrections.*

- `RealVector approxFnsPrevCenter`
  
  *copy of approxFnsCenter from the previous correction cycle*
8.116 SurrogateModel Class Reference

- RealMatrix approxGradsCenter
  unavailable when applying 1st-/2nd-order multiplicative corrections.

- RealVector truthFnsCenter
  Truth function values at the current correction point.

- RealVector truthFnsPrevCenter
  copy of truthFnsCenter from the previous correction cycle

- Variables subModelVars
  among differing variable views in force_rebuild()

- Constraints subModelCons
  among differing variable views in force_rebuild()

8.116.1 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.

8.116.2 Member Function Documentation

8.116.2.1 void compute_correction (const Response & truth_response, const Response & approx_response, const RealVector & c_vars) [protected, virtual]

agreement with truth_response

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.

Reimplemented from Model.

8.116.2.2 bool force_rebuild () [protected]

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.
8.116.3 Member Data Documentation

8.116.3.1 bool autoCorrection [protected]

and HierarchSurrModel approximate response computations
SurrBasedOptStrategy must toggle this value since compute_correction() no longer automatically backs out an old correction.

8.116.3.2 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().

The documentation for this class was generated from the following files:

- SurrogateModel.H
- SurrogateModel.C
8.117 SysCallAnalysisCode Class Reference

simulations using system calls.

Inheritance diagram for SysCallAnalysisCode::

```
AnalysisCode

SysCallAnalysisCode
```

Public Member Functions

- **SysCallAnalysisCode** (const ProblemDescDB &problem_db)
  
  *constructor*

- **~SysCallAnalysisCode** ()
  
  *destructor*

- void **spawn_evaluation** (const bool block_flag)
  
  *spawn a complete function evaluation*

- void **spawn_input_filter** (const bool block_flag)
  
  *spawn the input filter portion of a function evaluation*

- void **spawn_analysis** (const int &analysis_id, const bool block_flag)
  
  *spawn a single analysis as part of a function evaluation*

- void **spawn_output_filter** (const bool block_flag)
  
  *spawn the output filter portion of a function evaluation*

- const String & **command_usage** () const
  
  *return commandUsage*

Private Attributes

- String **commandUsage**
  
  *command syntax (supported only by SysCall analysis codes)*
8.117.1 Detailed Description

simulations using system calls.

SysCallAnalysisCode creates separate simulation processes using the C system() command. It utilizes CommandShell to manage shell syntax and asynchronous invocations.

8.117.2 Member Function Documentation

8.117.2.1 void spawn_evaluation (const bool block_flag)

spawn a complete function evaluation

Put the SysCallAnalysisCode to the shell using either the default syntax or specified commandUsage syntax. This function is used when all portions of the function evaluation (i.e., all analysis drivers) are executed on the local processor.

8.117.2.2 void spawn_input_filter (const bool block_flag)

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.

8.117.2.3 void spawn_analysis (const int & analysis_id, const bool block_flag)

spawn a single analysis as part of a function evaluation

Put a single analysis to the shell using the default syntax (no commandUsage support for analyses). 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.

8.117.2.4 void spawn_output_filter (const bool block_flag)

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.

The documentation for this class was generated from the following files:

- SysCallAnalysisCode.H
- SysCallAnalysisCode.C
8.118 SysCallApplicInterface Class Reference

using system calls.

Inheritance diagram for SysCallApplicInterface:

```
               Interface
                  ↓
ApplicationInterface
                  ↓
SysCallApplicInterface
```

Public Member Functions

- **SysCallApplicInterface** (const ProblemDescDB &problem_db)
  
  constructor

- **~SysCallApplicInterface** ()
  
  destructor

- void **derived_map** (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  
  that is specific to a derived class.

- void **derived_map_asynch** (const ParamResponsePair &pair)
  
  asynchronous evaluation that is specific to a derived class.

- void **derived_synch** (PRPList &prp_list)
- void **derived_synch_nowait** (PRPList &prp_list)
- int **derived_synchronous_local_analysis** (const int &analysis_id)
- const **StringArray** & **analysis_drivers** () const
  
  retrieve the analysis drivers specification for application interfaces

Private Member Functions

- void **spawn_application** (const bool block_flag)
  
  and output filter. Called from derived_map() & derived_map_asynch().

- void **derived_synch_kernel** (PRPList &prp_list)
  
  derived_synch_nowait()
• bool system_call_file_test (const String &root_file)
  
  the necessary results file(s)

Private Attributes

• SysCallAnalysisCode sysCallSimulator
  
  to a CommandShell in various combinations

• IntSet sysCallSet
  
  system call evaluations

• IntShortMap failCountMap
  
  map linking function evaluation id's to number of response read failures

8.118.1 Detailed Description

using system calls.

SysCallApplicInterface uses a SysCallAnalysisCode object for performing simulation invocations.

8.118.2 Member Function Documentation

8.118.2.1 void derived_synch (PRPList & prp_list) [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 ApplicationInterface.

8.118.2.2 void derived_synch_nowait (PRPList & prp_list) [inline, virtual]

Check for completion of active asynch jobs (tracked with sysCallSet). Make one pass through sysCallSet & complete all jobs that have returned.

Reimplemented from ApplicationInterface.
8.118.2.3  int derived_synchronous_local_analysis (const int & analysis_id)  [inline, virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().
Reimplemented from ApplicationInterface.

The documentation for this class was generated from the following files:

- SysCallApplicInterface.H
- SysCallApplicInterface.C
8.119  TANA3Approximation Class Reference

approximation (a multipoint approximation).

Inheritance diagram for TANA3Approximation::

```
Approximation

TANA3Approximation
```

Public Member Functions

- **TANA3Approximation ()**
  *default constructor*

- **TANA3Approximation (ProblemDescDB &problem_db, const size_t &num_vars)**
  *standard constructor*

- **~TANA3Approximation ()**
  *destructor*

Protected Member Functions

- **int num_coefficients () const**
  *derived class approximation type in numVars dimensions*

- **int num_constraints () const**
  *return the number of constraints to be enforced via anchorPoint*

- **void find_coefficients ()**
  *calculate the data fit coefficients using currentPoints and anchorPoint*

- **const Real & get_value (const RealVector &x)**
  *retrieve the approximate function value for a given parameter vector*

- **const RealBaseVector & get_gradient (const RealVector &x)**
  *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
  the vector of exponent values

- RealVector minX
  the vector of minimum parameter values used in scaling

- RealVector scX1
  the vector of scaled x1 values

- RealVector scX2
  the vector of scaled x2 values

- Real H
  the scalar Hessian value in the TANA-3 approximation

8.119.1 Detailed Description

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.

8.119.2 Member Function Documentation

8.119.2.1 void clear_current () [inline, protected, virtual]

Redefine default implementation to support history mechanism.

Reimplemented from Approximation.

The documentation for this class was generated from the following files:
- TANA3Approximation.H
- TANA3Approximation.C
8.120  TaylorApproximation Class Reference

series (a local approximation).

Inheritance diagram for TaylorApproximation::

```
Approximation
  /
TaylorApproximation
```

Public Member Functions

- TaylorApproximation ()
  default constructor

- TaylorApproximation (ProblemDescDB &problem_db, const size_t &num_vars)
  standard constructor

- ~TaylorApproximation ()
  destructor

Protected Member Functions

- int num_coefficients () const
  derived class approximation type in numVars dimensions

- void find_coefficients ()
  calculate the data fit coefficients using currentPoints and anchorPoint

- const Real & get_value (const RealVector &x)
  retrieve the approximate function value for a given parameter vector

- const RealBaseVector & get_gradient (const RealVector &x)
  retrieve the approximate function gradient for a given parameter vector

- const RealMatrix & get_hessian (const RealVector &x)
  retrieve the approximate function Hessian for a given parameter vector
8.120.1 Detailed Description

series (a local approximation).

The `TaylorApproximation` class provides a local approximation based on data from a single point in parameter space. It uses a first- or second-order Taylor series expansion: 
\[ f(x) = f(x_c) + \text{grad}(x_c)' (x - x_c) + (x - x_c)' \text{Hess}(x_c) (x - x_c) / 2. \]

The documentation for this class was generated from the following files:

- TaylorApproximation.H
- TaylorApproximation.C
8.121 Variables Class Reference

Base class for the variables class hierarchy.

Inheritance diagram for Variables::

```
Variables
    ├── AllVariables
    │    └── DistinctVariables
    │         └── MergedVariables
```

Public Member Functions

- **Variables ()**
  *default constructor*

- **Variables (const ProblemDescDB &problem_db)**
  *standard constructor*

- **Variables (const pair< short, short > &view)**
  *alternate constructor for instantiations on the fly*

- **Variables (const Variables &vars)**
  *copy constructor*

- virtual ```Variables ()```
  *destructor*

- **Variables operator= (const Variables &vars)**
  *assignment operator*

- virtual size_t tv () const
  *Returns total number of vars.*

- virtual const IntArray & merged_discrete_ids () const
  *returns the list of discrete variables merged into a continuous array*

- virtual const RealVector & continuous_variables () const
  *return the active continuous variables*

- virtual void continuous_variables (const RealVector &c_vars)
  *set the active continuous variables*
virtual const IntVector & discrete_variables () const
    return the active discrete variables

virtual void discrete_variables (const IntVector &d_vars)
    set the active discrete variables

virtual const StringArray & continuous_variable_labels () const
    return the active continuous variable labels

virtual void continuous_variable_labels (const StringArray &cv_labels)
    set the active continuous variable labels

virtual const StringArray & discrete_variable_labels () const
    return the active discrete variable labels

virtual void discrete_variable_labels (const StringArray &dv_labels)
    set the active discrete variable labels

virtual const RealVector & inactive_continuous_variables () const
    return the inactive continuous variables

virtual void inactive_continuous_variables (const RealVector &i_c_vars)
    set the inactive continuous variables

virtual const IntVector & inactive_discrete_variables () const
    return the inactive discrete variables

virtual void inactive_discrete_variables (const IntVector &i_d_vars)
    set the inactive discrete variables

virtual const StringArray & inactive_continuous_variable_labels () const
    return the inactive continuous variable labels

virtual void inactive_continuous_variable_labels (const StringArray &i_c_vars)
    set the inactive continuous variable labels

virtual const StringArray & inactive_discrete_variable_labels () const
    return the inactive discrete variable labels

virtual void inactive_discrete_variable_labels (const StringArray &i_d_vars)
    set the inactive discrete variable labels

virtual size_t acv () const
    returns total number of continuous vars
- virtual size_t adv() const  
  returns total number of discrete vars

- virtual RealVector all_continuous_variables() const  
  returns a single array with all continuous variables

- virtual void all_continuous_variables(const RealVector &a_c_vars)  
  sets all continuous variables using a single array

- virtual IntVector all_discrete_variables() const  
  returns a single array with all discrete variables

- virtual void all_discrete_variables(const IntVector &a_d_vars)  
  sets all discrete variables using a single array

- virtual StringArray all_continuous_variable_labels() const  
  returns a single array with all continuous variable labels

- virtual void all_continuous_variable_labels(const StringArray &a_c_v_labels)  
  sets all continuous variable labels using a single array

- virtual StringArray all_discrete_variable_labels() const  
  returns a single array with all discrete variable labels

- virtual void all_discrete_variable_labels(const StringArray &a_d_v_labels)  
  sets all discrete variable labels using a single array

- virtual StringArray all_variable_labels() const  
  returns a single array with all variable labels

- virtual void read(istream &s)  
  read a variables object from an istream

- virtual void write(ostream &s) const  
  write a variables object to an ostream

- virtual void write_aprepro(ostream &s) const  
  write a variables object to an ostream in aprepro format

- virtual void read.annotated(istream &s)  
  read a variables object in annotated format from an istream

- virtual void write.annotated(ostream &s) const  
  write a variables object in annotated format to an ostream

- virtual void write.tabular(ostream &s) const
write a variables object in tabular format to an 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 cv () const
  Returns number of active continuous vars.

- size_t dv () const
  Returns number of active discrete vars.

- size_t icv () const
  returns number of inactive continuous vars

- size_t idv () const
  returns number of inactive discrete vars

- Variables copy () const
  for use when a deep copy is needed (the representation is not shared)

- void reshape (const Sizet2DArray &vars_comps)
  variablesComponents

- const pair< short, short > & view () const
  returns variablesView

- pair< short, short > get_view (const ProblemDescDB &problem_db) const
  defines variablesView from problem_db attributes

- const String & variables_id () const
  returns the variables identifier string

- const Sizet2DArray & variables_components () const
  returns the number of variables for each of the constitutive components

- const StringArray & continuous_variable_types () const
  return the active continuous variable types
- const `StringArray & discrete_variable_types()` const
  return the active discrete variable types

- const `IntArray & continuous_variable_ids()` const
  return the active continuous variable position identifiers

- const `IntArray & inactive_continuous_variable_ids()` const
  return the inactive continuous variable position identifiers

- const `IntArray & all_continuous_variable_ids()` const
  return the all continuous variable position identifiers

- bool `is_null()` const
  function to check variablesRep (does this envelope contain a letter)

### Protected Member Functions

- `Variables (BaseConstructor, const ProblemDescDB &problem_db, const pair< short, short > &view)`
  derived class constructors - Coplien, p. 139)

- virtual void `copy_rep(const Variables &vars_rep)`
  Used by `copy()` to copy the contents of a letter class.

- virtual void `reshape_rep(const Sizet2DArray &vars_comps)`
  Used by `reshape()` to reshape the contents of a letter class.

### Protected Attributes

- `pair< short, short > variablesView`
  view enumerations

- `Sizet2DArray variablesComponents`
  design/uncertain/state (first index) by sub-type (second index)

- `StringArray continuousVarTypes`
  array of variable types for the active continuous variables

- `StringArray discreteVarTypes`
  array of variable types for the active discrete variables

- `IntArray continuousVarIds`
  array of position identifiers for the active continuous variables
- **IntArray inactiveContinuousVarIds**
  - array of position identifiers for the inactive continuous variables

- **IntArray allContinuousVarIds**
  - array of position identifiers for the all continuous variables array

- **RealVector emptyRealVector**
  - no variables corresponding to the request

- **IntVector emptyIntVector**
  - no variables corresponding to the request

- **StringArray emptyStringArray**
  - no variables corresponding to the request

### Private Member Functions

- **Variables * get_variables (const ProblemDescDB &problem_db)**
  - correct letter class

- **Variables * get_variables (const pair< short, short > &view) const**
  - and by copy() to instantiate a new letter class

### Private Attributes

- **String idVariables**
  - variables identifier string from the input file

- **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
8.121 Variables Class Reference

8.121.1 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.

8.121.2 Constructor & Destructor Documentation

8.121.2.1 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.

8.121.2.2 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.

8.121.2.3 Variables (const pair< short, short > & view)

alternate constructor for instantiations on the fly

This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to problem_db, the letter class is not fully populated. This constructor executes get_variables(view), which invokes the default constructor of the derived letter class, which in turn invokes the default constructor of the base class.

8.121.2.4 Variables (const Variables & vars)

copy constructor

Copy constructor manages sharing of variablesRep and incrementing of referenceCount.

8.121.2.5 ~Variables () [virtual]

destructor
Destructor decrements referenceCount and only deletes variablesRep when referenceCount reaches zero.

8.121.2.6 Variables (BaseConstructor, const ProblemDescDB & problem_db, const pair< short, short >& view) [protected]

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).

8.121.3 Member Function Documentation

8.121.3.1 Variables operator= (const Variables & vars)

assignment operator


8.121.3.2 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 bestVariables and data_pairs since these must catalogue copies (and should not change as the representation within currentVariables changes).

8.121.3.3 Variables * get_variables (const ProblemDescDB & problem_db) [private]

correct letter class

Initializes variablesRep to the appropriate derived type, as given by problem_db attributes. The standard derived class constructors are invoked.

8.121.3.4 Variables * get_variables (const pair< short, short > & view) const [private]

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.

8.121.4 Member Data Documentation
8.121.4.1 **IntArray continuousVarIds** [protected]

array of position identifiers for the active continuous variables
These identifiers define positions of the active continuous variables within the total variable sequence.

8.121.4.2 **IntArray inactiveContinuousVarIds** [protected]

array of position identifiers for the inactive continuous variables
These identifiers define positions of the inactive continuous variables within the total variable sequence.

8.121.4.3 **IntArray allContinuousVarIds** [protected]

array of position identifiers for the all continuous variables array
These identifiers define positions of the all continuous variables array within the total variable sequence.

The documentation for this class was generated from the following files:

- DakotaVariables.H
- DakotaVariables.C
8.122 VariablesUtil Class Reference

continuous and discrete variable domains.

Inheritance diagram for VariablesUtil:

```
VariablesUtil
  |         |         |
  |         |         |
  v         v         v
AllConstraints AllVariables DistinctConstraints DistinctVariables MergedConstraints MergedVariables
```

Public Member Functions

- **VariablesUtil ()**
  constructor

- **~VariablesUtil ()**
  destructor

Protected Member Functions

- void **update_merged** (const RealVector &c_array, const IntVector &d_array, RealVector &m_array) const
  array through promotion of integers to reals (merged view)

- void **update_all_continuous** (const RealVector &c1_array, const RealVector &c2_array, const RealVector &c3_array, RealVector &all_array) const
  continuous array (all view)

- void **update_all_discrete** (const IntVector &d1_array, const IntVector &d2_array, IntVector &all_array) const
  (all view)

- void **update_from_merged** (const RealVector &m_array, RealVector &c_array, IntVector &d_array) const
  array through truncation of reals to integers (merged view)

- void **update_from_all_continuous** (const RealVector &all_array, RealVector &c1_array, RealVector &c2_array, RealVector &c3_array) const
  continuous array (all view)

- void **update_from_all_discrete** (const IntVector &all_array, IntVector &d1_array, IntVector &d2_array) const
8.122 VariablesUtil Class Reference

(all view)

- **void update_labels** (const `StringArray` &l1_array, const `StringArray` &l2_array, const `StringArray` &l3_array, `StringArray` &all_array) const
  single label array (all view)

- **void update_labels** (const `StringArray` &l1_array, const `StringArray` &l2_array, `StringArray` &all_array) const
  combine 2 label arrays into a single label array (merged or all views)

- **void update_labels_partial** (size_t num_items, const `StringArray` &src_array, size_t src_start_index, `StringArray` &tgt_array, size_t tgt_start_index) const
  label array (all view)

- **void update_from_labels** (const `StringArray` &all_array, `StringArray` &l1_array, `StringArray` &l2_array, `StringArray` &l3_array)
  single label array (all view)

- **void update_from_labels** (const `StringArray` &all_array, `StringArray` &l1_array, `StringArray` &l2_array)
  extract 2 label arrays from a single label array (merged or all views)

8.122.1 Detailed Description

continuous and discrete variable domains.

Derived classes within the Variables and Constraints hierarchies use multiple inheritance to inherit these utilities.
The documentation for this class was generated from the following file:

- VariablesUtil.H
8.123  Vector Class Template Reference

Template class for the Dakota numerical vector.

Inheritance diagram for Vector::

```
BaseVector< T >

Vector
```

Public Member Functions

- **Vector ()**
  
  *Default constructor.*

- **Vector (size_t len)**
  
  *Constructor which takes an initial length.*

- **Vector (size_t len, const T &initial_val)**
  
  *Constructor which takes an initial length and an initial value.*

- **Vector (const Vector< T > &a)**
  
  *Copy constructor.*

- **Vector (const T *p, size_t len)**
  
  *Constructor which copies len entries from T.*

- **~Vector ()**
  
  *Destructor.*

- **Vector< T > & operator= (const Vector< T > &a)**
  
  *Normal const assignment operator.*

- **Vector< T > & operator= (const T &ival)**
  
  *Sets all elements in self to the value ival.*

- **operator T * () const**
  
  *Use with care!*

- **void read (istream &s)**
  
  *Reads a Vector from an input stream.*
- void read (istream &s, Array< String > &label_array)
  Reads a Vector and associated label array from an input stream.

- void read_partial (istream &s, size_t start_index, size_t num_items)
  Reads part of a Vector from an input stream.

- void read_partial (istream &s, size_t start_index, size_t num_items, Array< String > &label_array)
  Reads part of a Vector and the corresponding labels from an input stream.

- void read_tabular (istream &s)
  Reads a Vector from a tabular text input file.

- void read_annotated (istream &s, Array< String > &label_array)
  Reads a Vector from a tabular text input file.

- void write (ostream &s) const
  Writes a Vector to an output stream.

- void write (ostream &s, const Array< String > &label_array) const
  Writes a Vector and associated label array to an output stream.

- void write_partial (ostream &s, size_t start_index, size_t num_items) const
  Writes part of a Vector to an output stream.

- void write_partial (ostream &s, size_t start_index, size_t num_items, const Array< String > &label_array) const
  Writes part of a Vector to an output stream.

- void write_aprepro (ostream &s, const Array< String > &label_array) const
  Writes a Vector and associated label array to an output stream in aprepro format.

- void write_partial_aprepro (ostream &s, size_t start_index, size_t num_items, const Array< String > &label_array) const
  Writes part of a Vector to an output stream in aprepro format.

- void write_annotated (ostream &s, const Array< String > &label_array) const
  Writes a Vector to an output stream in aprepro format.

- void write_tabular (ostream &s) const
  Writes a Vector in tabular form to an output stream.

- void write_partial_tabular (ostream &s, size_t start_index, size_t num_items) const
  Writes part of a Vector in tabular form to an output stream.

- void read (BiStream &s, Array< String > &label_array)
Reads a Vector and associated label array from a binary input stream.

- void write (BoStream &s, const Array<String> &label_array) const
  Writes a Vector and associated label array to a binary output stream.

- void read (MPIUnpackBuffer &s)
  Reads a Vector from a buffer after an MPI receive.

- void read (MPIUnpackBuffer &s, Array<String> &label_array)
  MPI receive.

- void write (MPIPackBuffer &s) const
  Writes a Vector to a buffer prior to an MPI send.

- void write (MPIPackBuffer &s, const Array<String> &label_array) const
  an MPI send

8.123.1 Detailed Description

\texttt{template<class T> class Dakota::Vector<T>}

Template class for the Dakota numerical vector.

The Dakota::Vector class is the numeric vector class. It inherits from the common vector class Dakota::BaseVector which provides the same interface for both the STL and RW vector classes. If the STL version of BaseVector is based on the valarray class then some basic vector operations such as + , * are available. This class adds functionality to read/write vectors in a variety of ways

8.123.2 Constructor & Destructor Documentation

8.123.2.1 \textbf{Vector (const T * p, size_t len)} [inline]

Constructor which copies len entries from T*.
Assigns size values from p into array.

8.123.3 Member Function Documentation
8.123.3.1 **Vector< T > & operator= (const T & ival) [inline]**

Sets all elements in self to the value ival.

Assigns all values of array to ival. If STL, uses the vector assign method because there is no operator=(ival).

Reimplemented from **BaseVector**.

The documentation for this class was generated from the following file:

- DakotaVector.H
Chapter 9

DAKOTA File Documentation

9.1  JEGAOptimizer.C File Reference

Contains the implementation of the JEGAOptimizer class.

Namespaces

- namespace Dakota
- namespace std
- namespace JEGA::Logging
- namespace eddy::utilities

Classes

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

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

Functions

- template<typename T> string asstring (const T &val)
  Creates a string from the argument val using an ostringstream.
9.1.1 Detailed Description

Contains the implementation of the JEGAOptimizer class.
9.2 JEGAOptimizer.H File Reference

Contains the definition of the JEGAOptimizer class.

Namespaces

- namespace JEGA
- namespace JEGA::Utilities
- namespace JEGA::FrontEnd
- namespace JEGA::Algorithms
- namespace Dakota

Classes

- class JEGAOptimizer
  
  A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

9.2.1 Detailed Description

Contains the definition of the JEGAOptimizer class.
9.3  keywordtable.C File Reference

file containing keywords for the strategy, method, model, variables, interface, and responses input specifications from dakota.input.spec

Variables

- KeywordHandler idrKeywordTable [ ]
  
  KeywordHandler structure signifies the end of the keyword table.

9.3.1  Detailed Description

file containing keywords for the strategy, method, model, variables, interface, and responses input specifications from dakota.input.spec
9.4  main.C File Reference

file containing the main program for DAKOTA

Functions

- int main (int argc, char *argv[])

  The main DAKOTA program.

9.4.1  Detailed Description

file containing the main program for DAKOTA

9.4.2  Function Documentation

9.4.2.1  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.
9.5  restart_util.C File Reference

file containing the DAKOTA restart utility main program

Namespaces

- namespace 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 **main**(int argc, char *argv[])
  
  *The main program for the DAKOTA restart utility.*

9.5.1  Detailed Description

file containing the DAKOTA restart utility main program

9.5.2  Function Documentation
9.5.2.1 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()).
Chapter 10

Recommended Practices for DAKOTA Development

10.1 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. As a result, C++ coding styles are fairly standard across a variety of Sandia software projects in the engineering and computational sciences.

10.2 Style Guidelines

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

10.2.1 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.:

```c
int temporary_variable;
```

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

```c
const double CONSTANT_VALUE;
```

### 10.2.2 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
const RealVector& c_vars = model.continuous_variables(); // reference to continuousVariables cannot be changed
RealVector c_vars = model.continuous_variables(); // local copy of continuousVariables can be changed
```
10.2.3 Miscellaneous

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

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):

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.:

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

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

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

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

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

Lastly, #ifdef's are not indented (to make use of syntax highlighting in xemacs).
10.3 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.[CH], 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.H 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:

- .H A class header file ends in the suffix .H. 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.

- .C A class implementation file ends in the suffix .C. 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.

10.4 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 .C 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.
Chapter 11

Instructions for Modifying DAKOTA’s Input Specification

11.1 Modify dakota.input.spec

The master input specification resides in dakota.input.spec in Dakota/src. As part of the Input Deck Reader (IDR) build process, a soft link to this file is created in Dakota/packages/idr. The master input specification can be modified with the addition of new constructs using the following logical relationships:

- {} for required individual specifications
- () for required group specifications
- [] for optional individual specifications
- [] for optional group specifications
- | for "or" conditionals

These constructs can be used to define a variety of 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 constructs.

Warning:

- Do not skip this step. Attempts to modify the keywordtable.C and ProblemDescDB.C files in Dakota/src without reference to the results of the code 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.spec` are currently 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. [To allow keywords with capitalization and case sensitive parsing, IDR_NO_CONVRSN should be passed in `idr_init()` and uses of `idr_case_convert()` within `idr.c` should be reviewed.]

Since the Input Deck Reader (IDR) 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 keyword handler (the term "keyword handler" refers to the `strategy_kwhandler()`, `method_kwhandler()`, `variables_kwhandler()`, `interface_kwhandler()`, and `responses_kwhandler()` member functions in the `IDRProblemDescDB` class). 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.

Since IDR input is order-independent, the same keyword may be reused multiple times in the specification if and only if the specification blocks are mutually exclusive. For example, method selections (e.g., `dot_frcg`, `dot_bfgs`) can reuse the same method setting keywords (e.g., `optimization_type`) since the method selection blocks are all separated by logical "or"s. If `dot_frcg` and `dot_bfgs` were not exclusive and could be specified at the same time, then association of the `optimization_type` setting with a particular method would be ambiguous. This is the reason why repeated specifications which are non-exclusive must be made unique, typically with a prepended identifier (e.g., `cdv_initial_point`, `ddv_initial_point`).

### 11.2 Rebuild IDR

```
    cd Dakota/packages/idr/native
    make
```

These steps regenerate `keywordtable.C`, `idr-gen-code.C` and `idr-keyword.h` in the Dakota/packages/idr/native directory and copy the updated `keywordtable.C` to Dakota/src. As described in more detail in the next section, you must manually update `IDRProblemDescDB.C` in Dakota/src, based on `idr-gen-code.C`. If you commit changes to a source repository, be sure to commit the updated Dakota/packages/idr/native/idr-keyword.h, Dakota/src/dakota.input.spec, Dakota/src/keywordtable.C, and your manually updated Dakota/src/IDRProblemDescDB.C.

### 11.3 Update IDRProblemDescDB.C in Dakota/src

Find the keyword handler functions (e.g., `variables_kwhandler()` in Dakota/packages/idr/native/idr-gen-code.C and Dakota/src/IDRProblemDescDB.C which correspond to your modifications to the input specification. The `idr-gen-code.C` file is the result of a code generator and contains skeleton constructs for extracting data from IDR. You will be copying over parts of this skeleton to IDRProblemDescDB.C and then adding code to populate attributes within Data class container objects.

### 11.3.1 Replace keyword handler declarations and counter loop

Rather than trying to update these line by line, it is recommended to delete the entire block starting with the keyword declarations and ending at the bottom of the keyword counter loop. The declarations assign `-1` to keywords.
and look like this:

```c
Int cdv_descriptor = -1;
Int cdv_initial_point = -1;
```

They start after the line "Int cntr;". The keyword counter loop looks like this:

```c
for ( cntr=data_len; cntr--; ) {
    if ( idr_find_id( &cdv_descriptor, cntr,
        "cdv_descriptor", id_str, kw_str ) ) continue;
    ...
    if ( idr_find_id( &wuv_dist_upper_bounds, cntr,
        "wuv_dist_upper_bounds", id_str, kw_str ) ) continue;
}
```

Once the old keyword declarations and keyword counter loop have been deleted, replace them with the corresponding blocks from idr-gen-code.C containing the updated keyword declarations and counter loop.

### 11.3.2 Update keyword handler logic blocks

For the newly added or modified input specifications, copy the appropriate skeleton constructs from idr-gen-code.C and paste them into the corresponding location in IDRProblemDescDB.C.

The next step is to add code to these skeletons to set data attributes within the Data class object used by the keyword handler. At the top of the method, variables, interface, and responses keyword handlers, a Data class object is instantiated in order to store attributes, e.g.:

```c
DataMethod data_method;
```

and within the strategy keyword handler, a reference to the strategySpec data class object is used to store attributes. Each of these data class objects is a simple container class which contains the data from a single keyword handler invocation. Within each skeleton construct, you will extract data from the IDR data structures and then use this data to set the corresponding attribute within the Data class.

Integer, real, and string data are extracted using the idata, rdata, and cdata arrays provided by IDR. These arrays are indexed using a bracket operator with the keyword as an index. Lists of integer, list of real, and list of string data are extracted using the IDRProblemDescDB::idr_get_int_table(), IDRProblemDescDB::idr_get_real_table(), and IDRProblemDescDB::idr_get_string_table() functions, respectively.

**Example 1:** if you added the specification:

```c
[method_setting = <REAL>]
```

you would copy over

```c
    if ( method_setting >= 0 ) {
    }
```

from idr-gen-code.C into IDRProblemDescDB.C and then populate the if block with a call to set the corresponding attribute within the data_method object using data extracted using the rdata array:
if ( method_setting >= 0 ) {
    data_method.methodSetting = rdata[method_setting];
}

Use of a set member function within DataMethod is not needed since the data is public. The data is public since ProblemDescDB already provides sufficient encapsulation (ProblemDescDB::dataMethodList, ProblemDescDB::dataModelList, ProblemDescDB::dataVariablesList, ProblemDescDB::dataInterfaceList, ProblemDescDB::dataResponsesList, and ProblemDescDB::strategySpec are private attributes), and public access reduces the amount of code to manage when performing input specification modifications by omitting the need to add/modify set/get functions.

Example 2: if you added the specification

[method_setting = <LISTof><REAL>]

you would copy over

if ( method_setting >= 0 ) {
    Int idr_table_len;
    Real** idr_table = idr_get_real_table( parsed_data, method_setting, idr_table_len, 1, 1 );
}

from idr-gen-code.C into IDRProblemDescDB.C and then populate it with a loop which extracts each entry of the table and populates the corresponding attribute within the data_method object. The idr_table_len attribute is used for the loop limit and to size the data_method object.

if ( method_setting >= 0 ) {
    Int idr_table_len;
    Real** idr_table = idr_get_real_table( parsed_data, method_setting, idr_table_len, 1, 1 );

    data_method.methodSetting.reshape(idr_table_len);
    for (int i = 0; i<idr_table_len; i++)
        data_method.methodSetting[i] = idr_table[0][i];
}

Attention:
If no new data attributes have been added, but instead there are only new settings for existing attributes, then you’re done with the database augmentation at this point (you just need to add code to use these new settings in the places where the existing attributes are used).

11.4 Update ProblemDescDB.C in Dakota/src
11.4.1 Augment/update get_<data_type>() functions

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

```cpp
const RealVector& get_drv(const String& entry_name);
```

The implementation of each of these functions has a simple series of if-else checks which return the appropriate attribute based on the identifier string. For example,

```cpp
if (entry_name == "variables.continuous_design.initial_point")
return dbRep->dataVariablesIter->continuousDesignVars;
```

appears at the top of `ProblemDescDB::get_drv()`. Based on the identifier string, it returns the `continuous-DesignVars` attribute from a `DataVariables` object. Since there may be multiple variables specifications, the `dataVariablesIter` list iterator identifies which node in the list of `DataVariables` objects is used. In particular, `dataVariablesList` contains a list of all of the `data_variables` objects, one for each time `variables_kwhandler()` has been called by the parser. The particular variables object used for the data retrieval is managed by `dataVariablesIter`, 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_<data_type>()` functions, add `else` blocks with new identifier strings which retrieve 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:

```cpp
else if (entry_name == "method.method_name.method_setting")
return dbRep->dataMethodIter->methodSetting;
```

A strategy specification addition would not use a list iterator, and would instead look like:

```cpp
else if (entry_name == "strategy.strategy_name.strategy_setting")
return dbRep->strategySpec.strategySetting;
```

11.5 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 keyword handler logic blocks and Augment/update get_<data_type>() functions.
11.5.1 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).

11.5.2 Update the .C 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.

11.6 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.

11.7 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.spec that have been modified should be updated.
Chapter 12

Interfacing with DAKOTA as a Library

12.1 Introduction

Some users may be interested in linking the DAKOTA toolkit into another application for use as an algorithm library. While this is not the primary usage model for DAKOTA, certain facilities are in place to allow this type of integration.

As part of the normal DAKOTA build process, where `Dakota/configure -prefix=`pwd` has been run prior to `make` and `make install`, a `libdakota.a` is created and a copy of it is placed in `Dakota/lib`. This library contains all source files from `Dakota/src` excepting the `main.C` and `restart_util.C` main programs. This library may be linked with another application through inclusion of `-ldakota` on the link line. Library and header paths may also be specified using the `-L` and `-I` compiler options (using `Dakota/lib` and `Dakota/include`, respectively). 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++, SGOPT, LHS, Epetra, etc. Copies of these libraries are also placed in `Dakota/lib`. An XML specification of library names and paths is also available in `Dakota/dependency`.

Warning:

While users are free to interface DAKOTA as a library within other software applications for their own internal use, the GNU GPL license stipulates that any application linked with DAKOTA in this way defines a "derivative work" and can only be distributed externally under the same GNU GPL open source license. Refer to [http://www.gnu.org/licenses/gpl.html](http://www.gnu.org/licenses/gpl.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 DirectFnApplicInterface). 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, resulting in a "sandwich" implementation.
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.C, 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 running it (Strategy::run_strategy()). When using DAKOTA as an algorithm library, the operations are quite similar, although command line information (argc, argv, and therefore CommandLineHandler) will not in general be accessible. In particular, main.C can pass argc and argv into the ParallelLibrary and CommandLineHandler 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. That is, 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, 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).

With respect to alternate forms of ProblemDescDB::manage_inputs(), the two following sections describe different approaches to populating data within DAKOTA's problem description database. It is this database from which all DAKOTA objects draw data upon instantiation.

### 12.2 Problem database populated through 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 call to
12.3 Problem database populated through external means

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

should 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).

### 12.3 Problem database populated through external means

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 must be 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, a call to

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

will perform basic database error checking.

**Attention:**

The use of the `insert_node()` approach bypasses a considerable amount of data consistency enforcement within the keyword handler functions of `IDRProblemDescDB`. Application developers should study this logic and replicate as needed within their calling code in order to avoid run time errors resulting from data inconsistency. Abstraction of this consistency enforcement to the base `ProblemDescDB` level will simplify this process in the future.
12.4 Instantiating the strategy

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

```c++
// 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:

```c++
// 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.

12.5 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.

12.5.1 Extension

The first approach involves extending the existing DirectFnApplicInterface class to support additional direct simulation interfaces. In this case, a new simulation interface function can be added to Dakota/src/DirectFnApplic-Interface.[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:

```c++
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

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

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

This simulation can then be added to the logic blocks in DirectFnApplicInterface::derived_map_ac(). In addition, DirectFnApplicInterface::derived_map_if() and DirectFnApplicInterface::derived_map_of() can be extended to perform pre- and post-processing tasks if desired, but this is not required.

While this approach is the simplest, it has the disadvantage that the DAKOTA library 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.
12.5.2 Derivation

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

```cpp
namespace SIM {

class DirectFnApplicInterface: public Dakota::DirectFnApplicInterface {
    public:
        // Constructor and destructor
        DirectFnApplicInterface(const Dakota::ProblemDescDB& problem_db);
        ~DirectFnApplicInterface();

    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
    }
}
```

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

```cpp
// 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::DirectFnApplicInterface(problem_db), false);
```

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

```cpp
// retrieve the list of Models from the Strategy
ModelList& models = problem_db.model_list();
// iterate over the Model list
for (ModelLIter 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::DirectFnApplicInterface(problem_db), false);

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), DirectFnApplicInterface::analysisDrivers provides the analysis driver names specified by the user in the input file, and DirectFnApplicInterface::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.

### 12.6 Executing the strategy

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

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

### 12.7 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();
```

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

### 12.8 Summary

To utilize the DAKOTA library within a parent software application, the basic steps of main.C 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.C include the extension/derivation of the direct interface and the retrieval of strategy results after a run.

DAKOTA’s library mode has stabilized and is now being used successfully by several Sandia and external simulation codes/frameworks.
Chapter 13

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.

13.1 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.

13.2 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_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() 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.

### 13.3 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.
13.4 Software Tools for DAKOTA Development

13.4.1 Introduction

DAKOTA development relies on Subversion for revision control and the GNU Autotools for configuration management. This section lists these tools, where to acquire recommended versions, and how to configure them.

13.4.2 Subversion for Version Control

The DAKOTA project uses Subversion (http://subversion.tigris.org/) for software version control. To check DAKOTA out of the Subversion revision control system on development.sandia.gov, it may be necessary to install or upgrade the Subversion client on your system. We are presently using version 1.3.2 available from http://subversion.tigris.org/downloads/subversion-1.3.2.tar.gz.

To configure and build Subversion from source on your machine, the following settings should be used, since DAKOTA is hosted as a FSFS-type repository and depends on the external acro which is stored in a repository requiring SSL certificate handling:

```
tar xzf subversion-1.3.2.tar.gz
cd subversion-1.3.2
./configure --prefix=$HOME/local --with-ssl --without-berkeley-db CFLAGS=-O2
cd neon
./configure --prefix=$HOME/local --enable-shared --with-ssl --without-berkeley-db CFLAGS=-O2
cd ..
make && make check && make -k install
```

The make command as specified will ensure that Subversion is only installed if it passes all its self-tests, as well as making sure that the client install works correctly. Under some conditions, the Subversion build will attempt to write to /usr/lib, even when a --prefix option is passed to ./configure. This error may be disregarded when building the Subversion client, hence the --k option.

Once Subversion is working, DAKOTA (including externals) can be checked out with the single command

```
svn checkout svn+ssh://development.sandia.gov/usr/local/svn/Dakota/trunk Dakota
```

If you experience server timeouts when SVN attempts to fetch external packages through a proxy server, you might need to make a change to your $HOME/.subversion/servers file (generated for you the first time you run svn) by adding

```
[global]
http-proxy-exceptions = localhost, *.intranet.mydomain.com
http-proxy-host = wwwproxy.mydomain.com
```

to the bottom of the file. You should no longer get server timeouts when getting acro from software.sandia.gov. If you find that checking these three packages out from software is unacceptably slow, you may add your hostname
to the end of the http-proxy-exceptions line. Finally, svn will prompt you as to whether you wish to accept the
SSL certificate from software; type ‘p’ for permanent.
To set the default editor for Subversion commits, you may add the following to .cshrc:

```
setenv EDITOR "xemacs -g 81X50"
```

### 13.4.3 GNU Autotools for Configuration Management

DAKOTA uses the GNU Autotools ([http://www.gnu.org/software/autoconf/](http://www.gnu.org/software/autoconf/)) for configuration
management. Developers are currently using the following versions:

1. m4-1.4.3 ([http://ftp.gnu.org.gnu/m4/m4-1.4.3.tar.gz](http://ftp.gnu.org.gnu/m4/m4-1.4.3.tar.gz))

Building the tools in the order listed above should satisfy dependencies. For each PACKAGE the following build
process should suffice:

```
tar xzf $PACKAGE.tar.gz
cd $PACKAGE
./configure --prefix=$HOME/local
make
[make check]
make install
```

(Make check is useful for debugging builds of these packages, but optional and does take considerable time for
some packages.)
13.5  Todo List

Member Dakota::SurfpackApproximation::SurfpackApproximation(ProblemDescDB &problem_db, const size_t &num_acv)

The dakota data structures like RealVector inherit from std::vector.

Member Dakota::SurfpackApproximation::SurfpackApproximation(ProblemDescDB &problem_db, const size_t &num_acv)

Add RBFNet surface fit interface

Member Dakota::SurfpackApproximation::num_coefficients() const : Check to make sure that the number of points required does not

Member Dakota::SurfpackApproximation::num_coefficients() const : The reported number of points required is computed in a rather

Member Dakota::SurfpackApproximation::find_coefficients()  Right now, we’re completely deleting the old data and then

Member Dakota::SurfpackApproximation::get_hessian(const RealVector &x) Make this acceptably efficient

Member Dakota::SurfpackApproximation::checkForEqualityConstraints()  improve efficiency of conversion
Index

~Approximation
  Dakota::Approximation, 98
~BiStream
  Dakota::BiStream, 115
~Constraints
  Dakota::Constraints, 149
~EffGlobalOptimizer
  Dakota::EffGlobalOptimizer, 224
~Interface
  Dakota::Interface, 272
~Iterator
  Dakota::Iterator, 281
~Model
  Dakota::Model, 352
~OrthogonalPolynomial
  Dakota::OrthogonalPolynomial, 457
~ProblemDescDB
  Dakota::ProblemDescDB, 494
~Strategy
  Dakota::Strategy, 545
~Variables
  Dakota::Variables, 591
  _initPts
    Dakota::JEGAOptimizer, 295

A
  Dakota::CONMINOptimizer, 142
  accepts_multiple_points
    Dakota::JEGAOptimizer, 294
  actualModel
    Dakota::DataFitSurrModel, 161
  add_datapoint
    Dakota::Graphics, 251

AllConstraints
  Dakota::AllConstraints, 68
allContinuousVarIds
  Dakota::Variables, 593
AllVariables
  Dakota::AllVariables, 73
append_approximation
  Dakota::ApproximationInterface, 102
  Dakota::DataFitSurrModel, 159, 160
  approx_subprob_constraint_eval
    Dakota::SurrBasedOptStrategy, 563
  approx_subprob_objective_eval
    Dakota::SurrBasedOptStrategy, 563
  approxBuilds
    Dakota::SurrogateModel, 574
Approximation
  Dakota::Approximation, 97, 98
Array
  Dakota::Array, 106
  array
    Dakota::BaseVector, 112
  assign_rep
    Dakota::Interface, 273
    Dakota::Iterator, 283
    Dakota::Model, 355
asarstring
  Dakota, 58
asynchronous_local_analyses
  Dakota::ForkApplicInterface, 231
asynchronous_local_evaluations
  Dakota::ApplicationInterface, 91
asynchronous_local_evaluations_nowait
  Dakota::ApplicationInterface, 91
augmented_lagrangian_merit
  Dakota::SurrBasedOptStrategy, 562
autoCorrection
  Dakota::SurrogateModel, 574

B
  Dakota::CONMINOptimizer, 141
BaseVector
  Dakota::BaseVector, 110
begins
  Dakota::String, 549
BiStream
  Dakota::BiStream, 114, 115
BoStream
  Dakota::BoStream, 118, 119
BPA
INDEX 641

Dakota::NonDEvidence, 410
BPAC
  Dakota::NonDEvidence, 411
build_approximation
  Dakota::ApproximationInterface, 103
  Dakota::DataFitSurrModel, 159
build_global
  Dakota::DataFitSurrModel, 160
build_local_multipoint
  Dakota::DataFitSurrModel, 161

C
  Dakota::CONMINOptimizer, 141
cdf_beta_Pinv
  Dakota::NonD, 398
check_status
  Dakota::ForkAnalysisCode, 228
checkForEqualityConstraints
  Dakota::SurfpackApproximation, 552
clear_all
  Dakota::Approximation, 98
clear_current
  Dakota::Approximation, 98
  Dakota::TANA3Approximation, 581
close_streams
  Dakota::ParallelLibrary, 479
CMAX
  Dakota::NonDEvidence, 411
CMIN
  Dakota::NonDEvidence, 411
ColinPoint, 127
compute_correction
  Dakota::SurrogateModel, 573
concatenate_restart
  Dakota, 59
conminInfo
  Dakota::CONMINOptimizer, 139
constraint0_evaluator
  Dakota::SNLLOptimizer, 537
constraint1_evaluator
  Dakota::SNLLOptimizer, 537
constraint1_evaluator_gn
  Dakota::SNLLLeastSq, 530
constraint2_evaluator_gn
  Dakota::SNLLLeastSq, 531
constraint_violation
  Dakota::EffGlobalOptimizer, 225
  Dakota::SurrBasedOptStrategy, 563
constraintMappingIndices
  Dakota::CONMINOptimizer, 139
Dakota::DOTOptimizer, 219
constraintMappingMultipliers
  Dakota::CONMINOptimizer, 140
  Dakota::DOTOptimizer, 219
constraintMappingOffsets
  Dakota::CONMINOptimizer, 140
  Dakota::DOTOptimizer, 220
Constraints
  Dakota::Constraints, 149
constraintValues
  Dakota::CONMINOptimizer, 139
  Dakota::DOTOptimizer, 219
contains
  Dakota::List, 310
  Dakota::String, 548
continuousVarIds
  Dakota::Variables, 592
copy
  Dakota::Constraints, 150
  Dakota::Variables, 592
copy_results
  Dakota::ResponseRep, 518
count
  Dakota::List, 311
create_plots_2d
  Dakota::Graphics, 251
create_tabular_datastream
  Dakota::Graphics, 251
CreateEvaluator
  Dakota::JEGAOptimizer::EvaluatorCreator, 299
CT
  Dakota::CONMINOptimizer, 140
CtelRegexp, 152
DakFuncs0
  Dakota, 60
Dakota, 29
asstring, 58
concatenate_restart, 59
DakFuncs0, 60
eval_id_compare, 58
eval_id_sort_fn, 58
flush, 57
getdist, 57
getRmax, 58
mindist, 57
mindistindx, 58
operator==, 57
print_restart, 59
print_restart_tabular, 59
read_neutral, 59
repair_restart, 59
vars_set_compare, 58
Dakota::ActiveSet, 63
Dakota::ActiveSet
  derivVarsVector, 65
  requestVector, 65
Dakota::AllConstraints, 66
Dakota::AllConstraints
  AllConstraints, 68
Dakota::AllVariables, 70
Dakota::AllVariables
  AllVariables, 73
Dakota::AnalysisCode, 74
Dakota::Analyzer, 78
  evaluate_parameter_sets, 80
  print_vbd, 80
  var_based_decomp, 80
  volumetric_quality, 80
Dakota::ApplicationInterface, 82
Dakota::ApplicationInterface
  asynchronous_local_evaluations, 91
  asynchronous_local_evaluations_nowait, 91
  duplication_detect, 90
  init_serial, 88
  map, 88
  self_schedule_analyses, 89
  self_schedule_evaluations, 90
  serve_analyses_synch, 90
  serve_evaluations, 89
  serve_evaluations_async, 91
  serve_evaluations_peer, 92
  serve_evaluations_synch, 91
  static_schedule_evaluations, 90
  stop_evaluation_servers, 89
  synch, 88
  synch_nowait, 89
  synchronous_local_evaluations, 91
Dakota::Approximation, 93
  ~Approximation, 98
  Approximation, 97, 98
  clear_all, 98
  clear_current, 98
  get_approx, 99
  operator=, 98
Dakota::ApproximationInterface, 100
Dakota::ApproximationInterface
  append_approximation, 102
  build_approximation, 103
  functionSurfaces, 103
  update_approximation, 102
Dakota::Array, 104
  Array, 106
  data, 107
  operator T *, 106
  operator(), 107
  operator=, 106
  operator[], 107
Dakota::BaseConstructor, 108
Dakota::BaseVector, 109
Dakota::BaseVector
  array, 112
  BaseVector, 110
  data, 111
  length, 111
  operator(), 111
  operator[], 111
  reshape, 111
Dakota::BiStream, 113
Dakota::BiStream
  ~BiStream, 115
  BiStream, 114, 115
  operator>>, 115
Dakota::BoStream, 117
Dakota::BoStream
  BoStream, 118, 119
  operator<<, 119
Dakota::COLINAApplication, 120
  DoEval, 121
  map_response, 122
  next_eval, 122
  synchronize, 121
Dakota::COLINOptimizer, 123
  find_optimum, 125
  set_method_parameters, 125, 126
  set_runtime_parameters, 126
  set_standard_method_parameters, 125
Dakota::CommandLineHandler, 128
Dakota::CommandShell, 130
Dakota::CommandShell
  flush, 131
Dakota::ConcurrentStrategy, 132
Dakota::ConcurrentStrategy
  self_schedule_iterators, 133
  serve_iterators, 134
Dakota::CONMINOptimizer, 135
  A, 142
INDEX 643

B, 141
C, 141
conminInfo, 139
constraintMappingIndices, 139
constraintMappingMultipliers, 140
constraintMappingOffsets, 140
constraintValues, 139
CT, 140
DF, 141
G1, 141
G2, 141
IC, 142
ISC, 142
MS1, 141
N1, 140
N2, 140
N3, 140
N4, 140
N5, 140
optimizationType, 139
printControl, 139
S, 141
SCAL, 141
Dakota::Constraints, 143
~Constraints, 149
Constraints, 149
copy, 150
get_constraints, 150
manage_linear_constraints, 150
operator=, 150
reshape, 150
Dakota::DataFitSurrModel, 154
Dakota::DataFitSurrModel
actualModel, 161
append_approximation, 159, 160
build_approximation, 159
build_global, 160
build_local_multipoint, 161
derived_asynch_compute_response, 158
derived_compute_response, 158
derived_init_communicators, 160
derived_synchronize, 158
derived_synchronize_nowait, 158
evaluation_id, 160
update_actual_model, 161
update_approximation, 159
update_from_actual_model, 161
Dakota::DataInterface, 162
Dakota::DataMethod, 166

Dakota::DataModel, 177
Dakota::DataResponses, 180
Dakota::DataStrategy, 184
Dakota::DataVariables, 188
Dakota::DDACEDesignCompExp, 195
Dakota::DDACEDesignCompExp
DDACEDesignCompExp, 197
resolve_samples_symbols, 197
Dakota::DirectFnApplicInterface, 201
Dakota::DirectFnApplicInterface
derived_map_ac, 206
derived_synchronous_local_analysis, 206
Dakota::DistinctConstraints, 207
Dakota::DistinctConstraints
DistinctConstraints, 210
Dakota::DistinctVariables, 211
Dakota::DistinctVariables
DistinctVariables, 215
operator==, 215
Dakota::DOTOptimizer, 216
constraintMappingIndices, 219
constraintMappingMultipliers, 219
constraintMappingOffsets, 220
constraintValues, 219
dotFDSSinfo, 218
dotInfo, 218
dotMethod, 219
intCntlParmArray, 219
optimizationType, 219
printControl, 219
realCntlParmArray, 219
Dakota::EffGlobalOptimizer, 221
Dakota::EffGlobalOptimizer
~EffGlobalOptimizer, 224
constraint_violation, 225
objective, 224
Dakota::ForkAnalysisCode, 227
Dakota::ForkAnalysisCode
check_status, 228
Dakota::ForkApplicInterface, 229
Dakota::ForkApplicInterface
asynchronous_local_analyses, 231
derived_synchronous_local_analysis, 230
fork_application, 230
serve_analyses_asynch, 231
synchronous_local_analyses, 231
Dakota::FSUDesignCompExp, 232
Dakota::FSUDesignCompExp
enforce_input_rules, 234
FSUDesignCompExp, 234
Dakota::FunctionCompare, 236
Dakota::GaussProcApproximation, 237
Dakota::GaussProcApproximation
GPmodel_apply, 241
Dakota::GenLaguerreOrthogPolynomial, 243
Dakota::GetLongOpt, 245
Dakota::GetLongOpt
enroll, 247
GetLongOpt, 246
parse, 247
retrieve, 247
usage, 247
Dakota::Graphics, 249
add_datapoint, 251
create_plots_2d, 251
create_tabular_datastream, 251
new_dataset, 251
show_data_3d, 251
Dakota::GridApplicInterface, 253
Dakota::GridApplicInterface
derived_synchronous_local_analysis, 254
Dakota::HermiteOrthogPolynomial, 256
Dakota::HierarchSurrModel, 258
Dakota::HierarchSurrModel
derived_asynch_compute_response, 260
derived_compute_response, 260
derived_synchronize, 261
derived_synchronize_nowait, 261
evaluation_id, 261
Dakota::IDRProblemDescDB, 262
Dakota::IDRProblemDescDB
derived_manage_inputs, 264
Dakota::Interface, 265
~Interface, 272
assign_rep, 273
get_interface, 273
Interface, 272
operator=, 272
rawResponseArray, 273
rawResponseMap, 273
Dakota::Iterator, 275
~Iterator, 281
assign_rep, 283
derived_post_run, 284
derived_pre_run, 284
fdGradStepSize, 284
fdHessByFnStepSize, 285
fdHessByGradStepSize, 285
get_iterator, 284
Iterator, 281, 282
operator=, 282
post_run, 283
pre_run, 282, 283
print_results, 282
run, 282
run_iterator, 283
Dakota::JacobiOrthogPolynomial, 286
Dakota::JEGAOptimizer, 288
_initPts, 295
accepts_multiple_points, 294
find_optimum, 294
GetBestMOSolution, 293
GetBestSolution, 292
GetBestSOSolution, 293
initial_points, 294, 295
JEGAOptimizer, 290
LoadAlgorithmConfig, 291
LoadDakotaResponses, 291
LoadProblemConfig, 291
LoadTheConstraints, 292
LoadTheDesignVariables, 292
LoadTheObjectiveFunctions, 292
LoadTheParameterDatabase, 291
resize_response_results_array, 294
resize_variables_results_array, 293
returns_multiple_points, 294
ToDoubleMatrix, 293
Dakota::JEGAOptimizer::Driver, 296
DestroyAlgorithm, 297
Driver, 296
ExtractAllData, 297
PerformIterations, 297
Dakota::JEGAOptimizer::EvaluatorCreator, 299
Dakota::JEGAOptimizer::EvaluatorCreator
CreateEvaluator, 299
EvaluatorCreator, 299
Dakota::LaguerreOrthogPolynomial, 301
Dakota::LeastSq, 303
Dakota::LeastSq
derived_post_run, 305
LeastSq, 304
primary_resp_recast, 305
print_results, 305
read_observed_data, 305
run, 305
Dakota::LegendreOrthogPolynomial, 306
Dakota::List, 308
contains, 310
count, 311
find, 310
get, 309
index, 310, 311
insert, 310
operator[], 311
remove, 310
removeAt, 310
Dakota::Matrix, 312
operator=, 314
Dakota::MergedConstraints, 315
Dakota::MergedConstraints
MergedConstraints, 317
Dakota::MergedVariables, 318
Dakota::MergedVariables
MergedVariables, 321
Dakota::Minimizer, 322
initialize_scaling, 327
lin_coeffs_modify_n2s, 328
Minimizer, 327
modify_n2s, 328
modify_s2n, 328
response_modify_n2s, 328
secondary_resp_recast, 327
variables_recast, 327
Dakota::Model, 329
~Model, 352
assign_rep, 355
derivative_concurrency, 355
estimate_derivatives, 356
estimate_message_lengths, 355
get_model, 355
init_communicators, 354
init_serial, 355
interface, 353
interface_id, 354
local_eval_concurrency, 354
local_eval_synchronization, 354
manage_asv, 356
Model, 352
operator=, 353
subordinate_iterator, 353
subordinate_models, 354
surrogate_model, 353
synchronize_derivatives, 356
truth_model, 353
update_from_subordinate_model, 353
update_quasi_hessians, 356
update_response, 356
Dakota::MPIPackBuffer, 358
Dakota::MPIUnpackBuffer, 361
Dakota::MultilevelOptStrategy, 364
Dakota::MultilevelOptStrategy
run_coupled, 365
run_uncoupled, 366
run_uncoupled_adaptive, 366
Dakota::NCSUOptimizer, 367
NCSUOptimizer, 369
Dakota::NestedModel, 370
Dakota::NestedModel
derived_async_compute_response, 373
derived_compute_response, 373
derived_init_communicators, 374
derived_master_overload, 374
evaluation_id, 374
response_mapping, 374
subModel, 375
Dakota::Nl2Misc, 376
Dakota::NL2SOLLestSq, 377
Dakota::NLQLPOptimizer, 380
Dakota::NLSSOLLestSq, 385
Dakota::NoDBBaseConstructor, 387
Dakota::NonD, 388
Dakota::NonD
cdf_beta_Pinv, 398
hessian_d2X_dU2, 398
hessian_d2X_dZ2, 398
initialize_final_statistics, 395
initialize_random_variable_parameters, 395
initialize_random_variable_types, 395
initialize_random_variables, 394
jacobian_dU_dX, 398
jacobian_dX_dU, 397
jacobian_dX_dZ, 397
jacobian_dZ_dX, 398
Phi, 398
Phi_inverse, 398
trans_correlations, 396
trans_grad_U_to_X, 397
trans_grad_X_to_U, 397
trans_hess_X_to_U, 397
trans_U_to_X, 395
trans_U_to_Z, 395
trans_X_to_U, 396
trans_X_to_Z, 396
trans_Z_to_U, 396
trans_Z_to_X, 395
Dakota::NonDAdaptImpSampling, 400
Dakota::NonDAdaptImpSampling
  initialize, 402
Dakota::NonDCubature, 404
Dakota::NonDCubature
  NonDCubature, 405
  sampling_reset, 405
Dakota::NonDEvidence, 407
Dakota::NonDEvidence
  BPA, 410
  BPAC, 411
  CMAX, 411
  CMIN, 411
  IP, 411
  IPBEL, 411
  IPPLA, 411
  MAXINTVLS, 410
  NCMB, 410
  NI, 411
  NV, 410
  VMAX, 410
  VMIN, 410
  X, 411
  Y, 410
Dakota::NonDGlobalReliability, 413
Dakota::NonDIncremLHSSampling, 416
Dakota::NonDIncremLHSSampling
  NonDIncremLHSSampling, 417
  quantify_uncertainty, 417
Dakota::NonDIntegration, 419
Dakota::NonDIntegration
  NonDIntegration, 420
Dakota::NonDLHSSampling, 421
Dakota::NonDLHSSampling
  NonDLHSSampling, 422
  quantify_uncertainty, 423
Dakota::NonDLocalReliability, 424
Dakota::NonDLocalReliability
  dg_ds_eval, 429
  initial_taylor_series, 428
  initialize_class_data, 428
  initialize_level_data, 428
  initialize_mpp_search_data, 428
  probability, 429
  reliability, 429
  update_level_data, 429
  update_mpp_search_data, 429
  update_pma_reliability_level, 429
Dakota::NonDPolynomialChaos, 431
Dakota::NonDQuadrature, 433
Dakota::NonDQuadrature
  NonDQuadrature, 434
  sampling_reset, 434
Dakota::NonDReliability, 436
Dakota::NonDReliability
  jacobian_dX_dS, 439
  numerical_design_jacobian, 440
  PMA_constraint_eval, 439
  PMA_objective_eval, 439
  RIA_constraint_eval, 439
  RIA_objective_eval, 439
Dakota::NonDSampling, 441
Dakota::NonDSampling
  get_parameter_sets, 446
  NonDSampling, 445
  sampling_reset, 446
Dakota::NPSOLOptimizer, 447
  NPSOLOptimizer, 449
Dakota::Optimizer, 450
  derived_post_run, 453
  multi_objective_modify, 453
  multi_objective_retrieve, 453
  Optimizer, 452
  primary_resp_recast, 453
  print_results, 452
  run, 452
Dakota::OrthogonalPolynomial, 454
Dakota::OrthogonalPolynomial
  ~OrthogonalPolynomial, 457
  factorial, 457
  factorial_ratio, 457
  get_polynomial, 458
  n_choose_k, 457
  operator=, 457
  OrthogonalPolynomial, 456, 457
  pochhammer, 458
Dakota::OrthogPolyApproximation, 459
Dakota::OrthogPolyApproximation
  expectation, 464
  get_mean, 463
  get_mean_gradient, 463
  get_variance, 463
  gradient_check, 464
  integration, 463
  regression, 464
Dakota::ParallelConfiguration, 465
Dakota::ParallelLevel, 467
Dakota::ParallelLibrary, 470
Dakota::ParallelLibrary
  close_streams, 479
  increment_parallel_configuration, 479
  init_communicators, 479
  manage_outputs_restart, 479
ParallelLibrary, 478
  resolve_inputs, 479
  specify_outputs_restart, 478
Dakota::ParallelLibrary, 478
Dakota::ParamResponsePair, 481
Dakota::ParamResponsePair
  evalld, 484
  idInterface, 484
  ParamResponsePair, 483
  read, 483
  write, 484
Dakota::ParamResponsePair, 483
Dakota::ParamStudy, 485
Dakota::ProblemDescDB, 488
Dakota::ProblemDescDB
  ~ProblemDescDB, 494
  get_db, 495
  manage_inputs, 495
  operator=, 494
ProblemDescDB, 494
Dakota::PStudyDACE, 496
Dakota::PStudyDACE
  print_results, 498
  run, 497
Dakota::PStudyDACE, 497
Dakota::PSUADEDesignCompExp, 499
Dakota::PSUADEDesignCompExp
  enforce_input_rules, 501
  PSUADEDesignCompExp, 501
Dakota::RecastBaseConstructor, 502
Dakota::RecastModel, 503
Dakota::RecastModel
  initialize, 508
  RecastModel, 507
  update_from_sub_model, 508
Dakota::RecastModel, 507
Dakota::Response, 509
  Response, 513
Dakota::ResponseRep, 514
Dakota::ResponseRep
  copy_results, 518
  function_gradients, 519
  read, 517, 518
  read.annotated, 517
  read.tabular, 517
  reset, 519
  reset.inactive, 519
  reshape, 518
Dakota::ResponseRep, 516
  write, 517, 518
  write.annotated, 517
  write.tabular, 517
Dakota::SingleMethodStrategy, 520
Dakota::SingleModel, 522
Dakota::SNLLBase, 525
Dakota::SNLLLeastSq, 528
Dakota::SNLLLeastSq
  constraint1_evaluator_gn, 530
  constraint2_evaluator_gn, 531
  nlf2_evaluator_gn, 530
Dakota::SNLLOptimizer, 532
  constraint0_evaluator, 537
  constraint1_evaluator, 537
  nlf0_evaluator, 536
  nlf1_evaluator, 537
  nlf2_evaluator, 537
SNLLOptimizer, 536
Dakota::SOLBase, 539
Dakota::Strategy, 542
  ~Strategy, 545
  free_communicators, 546
  get_strategy, 546
  init_communicators, 546
  initialize_graphics, 546
  operator=, 545
  run_iterator, 545
Strategy, 544, 545
Dakota::String, 547
  begins, 549
  contains, 548
  data, 549
  ends, 549
  lower, 548
  operator const char *, 548
  upper, 548
Dakota::SurfpackApproximation, 550
Dakota::SurfpackApproximation
  checkForEqualityConstraints, 552
  find_coefficients, 551
  get_hessian, 552
SurfpackApproximation, 551
  surrogates_to_surf_data, 552
Dakota::SurrBasedOptStrategy, 554
Dakota::SurrBasedOptStrategy
  approx_subprob_constraint_eval, 563
  approx_subprob_objective_eval, 563
  augmented_lagrangian_merit, 562

DAKOTA Version 4.1 Developers Manual generated on October 11, 2007
constraint_violation, 563
hard_convergence_check, 561
hom_constraint_eval, 564
hom_objective_eval, 564
lagrangian_merit, 562
objective, 563
objective_gradient, 563
penalty_merit, 563
run_strategy, 561
tr_ratio_check, 561
update_augmented_lagrange_multipliers, 562
update_filter, 562
update_lagrange_multipliers, 562
update_penalty, 562
Dakota::SurrogateDataPoint, 565
Dakota::SurrogateDataPointRep, 567
Dakota::SurrogateModel, 569
Dakota::SurrogateModel
approxBuilds, 574
autoCorrection, 574
compute_correction, 573
force_rebuild, 573
Dakota::SysCallAnalysisCode, 575
Dakota::SysCallAnalysisCode
spawn_analysis, 576
spawn_evaluation, 576
spawn_input_filter, 576
spawn_output_filter, 576
Dakota::SysCallApplicInterface, 577
Dakota::SysCallApplicInterface
derived_synch, 578
derived_synch_nowait, 578
derived_synchronous_local_analysis, 578
Dakota::TANA3Approximation, 580
clear_current, 581
Dakota::TaylorApproximation, 583
Dakota::Variables, 585
~Variables, 591
allContinuousVarIds, 593
continuousVarIds, 592
copy, 592
get_variables, 592
inactiveContinuousVarIds, 593
operator=, 592
Variables, 591, 592
Dakota::VariablesUtil, 594
Dakota::Vector, 596
operator=, 598
Vector, 598
data
Dakota::Array, 107
Dakota::BaseVector, 111
Dakota::String, 549
DDACE\textsuperscript{Design}CompExp
Dakota::DDACE\textsuperscript{Design}CompExp, 197
derivative_concurrency
Dakota::Model, 355
derived_asynch_compute_response
Dakota::DataFitSurrModel, 158
Dakota::HierarchSurrModel, 260
Dakota::NestedModel, 373
derived_compute_response
Dakota::DataFitSurrModel, 158
Dakota::HierarchSurrModel, 260
Dakota::NestedModel, 373
derived_manage_inputs
Dakota::IDRProblemDescDB, 264
derived_map_ac
Dakota::DirectFnApplicInterface, 206
derived_master_overload
Dakota::NestedModel, 374
derived_post_run
Dakota::Iterator, 284
Dakota::LeastSq, 305
Dakota::Optimizer, 453
derived_pre_run
Dakota::Iterator, 284
derived_synch
Dakota::SysCallApplicInterface, 578
derived_synch_nowait
Dakota::SysCallApplicInterface, 578
derived_synchronize
Dakota::DataFitSurrModel, 158
Dakota::HierarchSurrModel, 261
derived_synchronize_nowait
Dakota::DataFitSurrModel, 158
Dakota::HierarchSurrModel, 261
derived_synchronous_local_analysis
Dakota::DirectFnApplicInterface, 206
Dakota::ForkApplicInterface, 230
Dakota::GridApplicInterface, 254
Dakota::SysCallApplicInterface, 578
derivVarsVector
Dakota::ActiveSet, 65
DestroyAlgorithm
Dakota::JEGAOptimizer::Driver, 297
DF
   Dakota::CONMINOptimizer, 141
dg_ds_eval
   Dakota::NonDLocalReliability, 429
DistinctConstraints
   Dakota::DistinctConstraints, 210
DistinctVariables
   Dakota::DistinctVariables, 215
DoEval
   Dakota::COLINApplication, 121
dotFDSinfo
   Dakota::DOTOptimizer, 218
dotInfo
   Dakota::DOTOptimizer, 218
dotMethod
   Dakota::DOTOptimizer, 219
Driver
   Dakota::JEGAOptimizer::Driver, 296
duplication_detect
   Dakota::ApplicationInterface, 90
ends
   Dakota::String, 549
enforce_input_rules
   Dakota::FSUDesignCompExp, 234
   Dakota::PSUADEDesignCompExp, 501
enroll
   Dakota::GetLongOpt, 247
ErrorTable, 226
estimate_derivatives
   Dakota::Model, 356
estimate_message_lengths
   Dakota::Model, 355
eval_id_compare
   Dakota::ParamResponsePair, 484
eval_id_sort_fn
   Dakota::ParamResponsePair, 484
evalId
   Dakota::ParamResponsePair, 484
evaluate_parameter_sets
   Dakota::Analyzer, 80
evalluation_id
   Dakota::DataFitSurrModel, 160
   Dakota::HierarchSurrModel, 261
   Dakota::NestedModel, 374
EvaluatorCreator
   Dakota::JEGAOptimizer::EvaluatorCreator, 299
expectation
   Dakota::OrthogPolyApproximation, 464
ExtractAllData
   Dakota::JEGAOptimizer::Driver, 297
factorial
   Dakota::OrthogonalPolynomial, 457
factorial_ratio
   Dakota::OrthogonalPolynomial, 457
fdGradStepSize
   Dakota::Iterator, 284
fdHessByFnStepSize
   Dakota::Iterator, 285
fdHessByGradStepSize
   Dakota::Iterator, 285
find
   Dakota::List, 310
find_coefficients
   Dakota::SurfpackApproximation, 551
find_optimum
   Dakota::COLINOptimizer, 125
   Dakota::JEGAOptimizer, 294
flush
   Dakota, 57
   Dakota::CommandShell, 131
force_rebuild
   Dakota::SurrogateModel, 573
fork_application
   Dakota::ForkApplicInterface, 230
free_communicators
   Dakota::Strategy, 546
FSUDesignCompExp
   Dakota::FSUDesignCompExp, 234
functionGradients
   Dakota::ResponseRep, 519
functionSurfaces
   Dakota::ApproximationInterface, 103
G1
   Dakota::CONMINOptimizer, 141
G2
   Dakota::CONMINOptimizer, 141
get
   Dakota::List, 309
get_approx
   Dakota::Approximation, 99
get_constraints
   Dakota::Constraints, 150
get_db
   Dakota::ProblemDescDB, 495
get_hessian
   Dakota::OrthogPolyApproximation, 464
Dakota::SurfpackApproximation, 552
get_interface
    Dakota::Interface, 273
get_iterator
    Dakota::Iterator, 284
get_mean
    Dakota::OrthogPolyApproximation, 463
get_mean_gradient
    Dakota::OrthogPolyApproximation, 463
get_model
    Dakota::Model, 355
get_parameter_sets
    Dakota::NonDSampling, 446
get_polynomial
    Dakota::OrthogonalPolynomial, 458
get_strategy
    Dakota::Strategy, 546
get_variables
    Dakota::Variables, 592
get_variance
    Dakota::OrthogPolyApproximation, 463
GetBestMOSolution
    Dakota::JEGAOptimizer, 293
GetBestSolution
    Dakota::JEGAOptimizer, 292
GetBestSOSolution
    Dakota::JEGAOptimizer, 293
getdist
    Dakota, 57
GetLongOpt
    Dakota::GetLongOpt, 246
getRmax
    Dakota, 58
GPmodel_apply
    Dakota::GaussProcApproximation, 241
gradient_check
    Dakota::OrthogPolyApproximation, 464
hard_convergence_check
    Dakota::SurrBasedOptStrategy, 561
hessian_d2X_dU2
    Dakota::NonD, 398
hessian_d2X_dZ2
    Dakota::NonD, 398
hom_constraint_eval
    Dakota::SurrBasedOptStrategy, 564
hom_objective_eval
    Dakota::SurrBasedOptStrategy, 564
IC
Dakota::CONMINOptimizer, 142
idInterface
    Dakota::ParamResponsePair, 484
inactiveContinuousVarIds
    Dakota::Variables, 593
increment_parallel_configuration
    Dakota::ParallelLibrary, 479
index
    Dakota::List, 310, 311
init_communicators
    Dakota::Model, 354
    Dakota::ParallelLibrary, 479
    Dakota::Strategy, 546
init_serial
    Dakota::ApplicationInterface, 88
    Dakota::Model, 355
initial_points
    Dakota::JEGAOptimizer, 294, 295
initial_taylor_series
    Dakota::NonDLocalReliability, 428
initialize
    Dakota::NonDAdaptImpSampling, 402
    Dakota::RecastModel, 508
initialize_class_data
    Dakota::NonDLocalReliability, 428
initialize_final_statistics
    Dakota::NonD, 395
initialize_graphics
    Dakota::Strategy, 546
initialize_level_data
    Dakota::NonDLocalReliability, 428
initialize_mpp_search_data
    Dakota::NonDLocalReliability, 428
initialize_random_variable_parameters
    Dakota::NonD, 395
initialize_random_variable_types
    Dakota::NonD, 395
initialize_random_variables
    Dakota::NonD, 394
initialize_scaling
    Dakota::Minimizer, 327
insert
    Dakota::List, 310
intCntrlParamArray
    Dakota::DOTOptimizer, 219
integration
    Dakota::OrthogPolyApproximation, 463
Interface
    Dakota::Interface, 272
<table>
<thead>
<tr>
<th>Term</th>
<th>Class/Module</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface</td>
<td>Dakota::Model</td>
<td>353</td>
</tr>
<tr>
<td>interface_id</td>
<td>Dakota::Model</td>
<td>354</td>
</tr>
<tr>
<td>IP</td>
<td>Dakota::NonDEvidence</td>
<td>411</td>
</tr>
<tr>
<td>IPBEL</td>
<td>Dakota::NonDEvidence</td>
<td>411</td>
</tr>
<tr>
<td>IPPLA</td>
<td>Dakota::NonDEvidence</td>
<td>411</td>
</tr>
<tr>
<td>ISC</td>
<td>Dakota::CONMINOptimizer</td>
<td>142</td>
</tr>
<tr>
<td>Iterator</td>
<td>Dakota::Iterator</td>
<td>281, 282</td>
</tr>
<tr>
<td>jacobian_dU_dX</td>
<td>Dakota::NonD</td>
<td>398</td>
</tr>
<tr>
<td>jacobian_dX_dS</td>
<td>Dakota::NonDEReliability</td>
<td>439</td>
</tr>
<tr>
<td>jacobian_dX_dU</td>
<td>Dakota::NonD</td>
<td>397</td>
</tr>
<tr>
<td>jacobian_dX_dZ</td>
<td>Dakota::NonD</td>
<td>397</td>
</tr>
<tr>
<td>jacobian_dZ_dX</td>
<td>Dakota::NonD</td>
<td>398</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>Dakota::JEGAOptimizer</td>
<td>290</td>
</tr>
<tr>
<td>JEGAOptimizer.C</td>
<td>Dakota::JEGAOptimizer</td>
<td>601</td>
</tr>
<tr>
<td>JEGAOptimizer.H</td>
<td>Dakota::JEGAOptimizer</td>
<td>603</td>
</tr>
<tr>
<td>keywordtable.C</td>
<td>Dakota::JEGAOptimizer</td>
<td>604</td>
</tr>
<tr>
<td>lagrangian_merit</td>
<td>Dakota::SurrBasedOptStrategy</td>
<td>562</td>
</tr>
<tr>
<td>LeastSq</td>
<td>Dakota::LeastSq</td>
<td>304</td>
</tr>
<tr>
<td>length</td>
<td>Dakota::BaseVector</td>
<td>111</td>
</tr>
<tr>
<td>lin_coeffs_modify_n2s</td>
<td>Dakota::Minimizer</td>
<td>328</td>
</tr>
<tr>
<td>LoadAlgorithmConfig</td>
<td>Dakota::JEGAOptimizer</td>
<td>291</td>
</tr>
<tr>
<td>LoadDakotaResponses</td>
<td>Dakota::JEGAOptimizer</td>
<td>291</td>
</tr>
<tr>
<td>LoadProblemConfig</td>
<td>Dakota::JEGAOptimizer</td>
<td>291</td>
</tr>
<tr>
<td>LoadTheConstraints</td>
<td>Dakota::JEGAOptimizer</td>
<td>292</td>
</tr>
<tr>
<td>LoadTheDesignVariables</td>
<td>Dakota::JEGAOptimizer</td>
<td>292</td>
</tr>
<tr>
<td>LoadTheObjectiveFunctions</td>
<td>Dakota::JEGAOptimizer</td>
<td>292</td>
</tr>
<tr>
<td>LoadTheParameterDatabase</td>
<td>Dakota::JEGAOptimizer</td>
<td>291</td>
</tr>
<tr>
<td>local_eval_concurrency</td>
<td>Dakota::Model</td>
<td>354</td>
</tr>
<tr>
<td>local_eval_synchronization</td>
<td>Dakota::Model</td>
<td>354</td>
</tr>
<tr>
<td>lower</td>
<td>Dakota::String</td>
<td>548</td>
</tr>
<tr>
<td>main</td>
<td>Dakota::JEGAOptimizer</td>
<td>292</td>
</tr>
<tr>
<td>main.C</td>
<td>Dakota::JEGAOptimizer</td>
<td>605</td>
</tr>
<tr>
<td>restart_util.C</td>
<td>Dakota::JEGAOptimizer</td>
<td>606</td>
</tr>
<tr>
<td>manage_asv</td>
<td>Dakota::Model</td>
<td>356</td>
</tr>
<tr>
<td>manage_inputs</td>
<td>Dakota::ProblemDescDB</td>
<td>495</td>
</tr>
<tr>
<td>manage_linear_constraints</td>
<td>Dakota::Constraints</td>
<td>150</td>
</tr>
<tr>
<td>manage_outputs_restart</td>
<td>Dakota::ParallelLibrary</td>
<td>479</td>
</tr>
<tr>
<td>map</td>
<td>Dakota::ApplicationInterface</td>
<td>88</td>
</tr>
<tr>
<td>map_response</td>
<td>Dakota::COLINApplication</td>
<td>122</td>
</tr>
<tr>
<td>MAXINTVLS</td>
<td>Dakota::NonDEvidence</td>
<td>410</td>
</tr>
<tr>
<td>MergedConstraints</td>
<td>Dakota::MergedConstraints</td>
<td>317</td>
</tr>
<tr>
<td>MergedVariables</td>
<td>Dakota::MergedVariables</td>
<td>321</td>
</tr>
<tr>
<td>mindist</td>
<td>Dakota::SurrBasedOptStrategy</td>
<td>562</td>
</tr>
<tr>
<td>mindistindx</td>
<td>Dakota::SurrBasedOptStrategy</td>
<td>562</td>
</tr>
<tr>
<td>Minimizer</td>
<td>Dakota::Minimizer</td>
<td>327</td>
</tr>
<tr>
<td>Model</td>
<td>Dakota::Model</td>
<td>352</td>
</tr>
<tr>
<td>modify_n2s</td>
<td>Dakota::Minimizer</td>
<td>328</td>
</tr>
<tr>
<td>modify_s2n</td>
<td>Dakota::Minimizer</td>
<td>328</td>
</tr>
<tr>
<td>MS1</td>
<td>Dakota::CONMINOptimizer</td>
<td>141</td>
</tr>
<tr>
<td>multi_objective_modify</td>
<td>Dakota::Minimizer</td>
<td>453</td>
</tr>
</tbody>
</table>
multi_objective_retrieve
   Dakota::Optimizer, 453

N1
   Dakota::CONMINOptimizer, 140
N2
   Dakota::CONMINOptimizer, 140
N3
   Dakota::CONMINOptimizer, 140
N4
   Dakota::CONMINOptimizer, 140
N5
   Dakota::CONMINOptimizer, 140
n_choose_k
   Dakota::OrthogonalPolynomial, 457

NCMB
   Dakota::NonDEvidence, 410

NCSUOptimizer
   Dakota::NCSUOptimizer, 369

new_dataset
   Dakota::Graphics, 251

next_eval
   Dakota::COLINAApplication, 122

NI
   Dakota::NonDEvidence, 411

nlf0_evaluator
   Dakota::SNLLOptimizer, 536

nlf1_evaluator
   Dakota::SNLLOptimizer, 537

nlf2_evaluator
   Dakota::SNLLOptimizer, 537

nlf2_evaluator_gn
   Dakota::SNLLLeastSq, 530

NonDCubature
   Dakota::NonDCubature, 405

NonDIncremLHSSampling
   Dakota::NonDIncremLHSSampling, 417

NonDIteration
   Dakota::NonDIteration, 420

NonDLHSSampling
   Dakota::NonDLHSSampling, 422

NonDQuadrature
   Dakota::NonDQuadrature, 434

NonDSampling
   Dakota::NonDSampling, 445

NPSOLOptimizer
   Dakota::NPSOLOptimizer, 449

numerical_design_jacobian
   Dakota::NonDReliability, 440

NV
   Dakota::NonDEvidence, 410

objective
   Dakota::EffGlobalOptimizer, 224
   Dakota::SurrBasedOptStrategy, 563

objective_gradient
   Dakota::SurrBasedOptStrategy, 563

operator const char *
   Dakota::String, 548

operator T *
   Dakota::Array, 106

operator()
   Dakota::Array, 107
   Dakota::BaseVector, 111

operator<<
   Dakota::BoStream, 119

operator=
   Dakota::Approximation, 98
   Dakota::Array, 106
   Dakota::Constraints, 150
   Dakota::Interface, 272
   Dakota::Iterator, 282
   Dakota::Matrix, 314
   Dakota::Model, 353
   Dakota::OrthogonalPolynomial, 457
   Dakota::ProblemDescDB, 494
   Dakota::Strategy, 545
   Dakota::Variables, 592
   Dakota::Vector, 598

operator==
   Dakota, 57
   Dakota::DistinctVariables, 215

operator>>
   Dakota::BiStream, 115

operator[]
   Dakota::Array, 107
   Dakota::BaseVector, 111
   Dakota::List, 311

optimizationType
   Dakota::CONMINOptimizer, 139
   Dakota::DOTOptimizer, 219

Optimizer
   Dakota::Optimizer, 452

OrthogonalPolynomial
   Dakota::OrthogonalPolynomial, 456, 457

ParallelLibrary
   Dakota::ParallelLibrary, 478

ParamResponsePair
   Dakota::ParamResponsePair, 483
response_modify_n2s
  Dakota::Minimizer, 328
ResponseRep
  Dakota::ResponseRep, 516
restart_util.C, 606
  main, 606
retrieve
  Dakota::GetLongOpt, 247
returns_multiple_points
  Dakota::JEGAOptimizer, 294
RIA_constraint_eval
  Dakota::NonDReliability, 439
RIA_objective_eval
  Dakota::NonDReliability, 439
run
  Dakota::LeastSq, 305
  Dakota::Optimizer, 452
  Dakota::PStudyDACE, 497
run_coupled
  Dakota::MultilevelOptStrategy, 365
run_iteror
  Dakota::Strategy, 545
run_strategy
  Dakota::SurrBasedOptStrategy, 561
run_uncoupled
  Dakota::MultilevelOptStrategy, 366
run_uncoupled_adaptive
  Dakota::MultilevelOptStrategy, 366
S
  Dakota::CONMINOptimizer, 141
sampling_reset
  Dakota::NonDCubature, 405
  Dakota::NonDQuadrature, 434
  Dakota::NonDSampling, 446
SCAL
  Dakota::CONMINOptimizer, 141
secondary_resp_recast
  Dakota::Minimizer, 327
self_schedule_analyses
  Dakota::ApplicationInterface, 89
self_schedule_evaluations
  Dakota::ApplicationInterface, 90
self_schedule_iterators
  Dakota::ApplicationInterface, 90
serve_analyses
serve_analyses_async
  Dakota::ApplicationInterface, 90
serve_analyses_sync
  Dakota::ApplicationInterface, 91
serve_evaluations
  Dakota::ApplicationInterface, 89
serve_evaluations_async
  Dakota::ApplicationInterface, 91
serve_evaluations_peer
  Dakota::ApplicationInterface, 92
serve_evaluations_synchron
  Dakota::ApplicationInterface, 91
serve_iterators
  Dakota::ConcurrentStrategy, 134
set_method_parameters
  Dakota::COLINOptimizer, 125, 126
set_runtime_parameters
  Dakota::COLINOptimizer, 126
set_standard_method_parameters
  Dakota::COLINOptimizer, 125
show_data_3d
  Dakota::Graphics, 251
SIM, 61
SIM::DirectFnApplicInterface, 199
SNLLOptimizer
  Dakota::SNLLOptimizer, 536
spawn_analysis
  Dakota::SysCallAnalysisCode, 576
spawn_evaluation
  Dakota::SysCallAnalysisCode, 576
spawn_input_filter
  Dakota::SysCallAnalysisCode, 576
spawn_output_filter
  Dakota::SysCallAnalysisCode, 576
specify_outputs_restart
  Dakota::ParallelLibrary, 478
static_schedule_evaluations
  Dakota::ApplicationInterface, 90
stop_evaluation_servers
  Dakota::ApplicationInterface, 89
Strategy
  Dakota::Strategy, 544, 545
subModel
  Dakota::NestedModel, 375
subordinate_iterator
  Dakota::Model, 353
subordinate_models
  Dakota::Model, 354
SurfpackApproximation
  Dakota::SurfpackApproximation, 551
surrogate_model
  Dakota::Model, 353
surrogates_to_surf_data
   Dakota::SurfpackApproximation, 552

sync
   Dakota::ApplicationInterface, 88

sync_nowait
   Dakota::ApplicationInterface, 89

synchronize
   Dakota::COLINApplication, 121

synchronize_derivatives
   Dakota::Model, 356

synchronous_local_analyses
   Dakota::ForkApplicInterface, 231

synchronous_local_evaluations
   Dakota::ApplicationInterface, 91

ToDoubleMatrix
   Dakota::JEGAOptimizer, 293

tr_ratio_check
   Dakota::SurrBasedOptStrategy, 561

trans_correlations
   Dakota::NonD, 396

trans_grad_U_to_X
   Dakota::NonD, 397

trans_grad_X_to_U
   Dakota::NonD, 397

trans_hess_X_to_U
   Dakota::NonD, 397

trans_U_to_X
   Dakota::NonD, 395

trans_U_to_Z
   Dakota::NonD, 395

trans_X_to_U
   Dakota::NonD, 396

trans_X_to_Z
   Dakota::NonD, 396

trans_Z_to_U
   Dakota::NonD, 396

trans_Z_to_X
   Dakota::NonD, 395

truth_model
   Dakota::Model, 353

update_actual_model
   Dakota::DataFitSurrModel, 161

update_approximation
   Dakota::ApproximationInterface, 102

update_augmented_lagrange_multipliers
   Dakota::SurrBasedOptStrategy, 562

update_filter
   Dakota::SurrBasedOptStrategy, 562

update_from_actual_model
   Dakota::DataFitSurrModel, 161

update_from_sub_model
   Dakota::RecastModel, 508

update_from_subordinate_model
   Dakota::Model, 353

update_lagrange_multipliers
   Dakota::SurrBasedOptStrategy, 562

update_level_data
   Dakota::NonDLocalReliability, 429

update_mpp_search_data
   Dakota::NonDLocalReliability, 429

update_penalty
   Dakota::SurrBasedOptStrategy, 562

update_pma_reliability_level
   Dakota::NonDLocalReliability, 429

update_quasi_hessians
   Dakota::Model, 356

update_response
   Dakota::Model, 356

upper
   Dakota::String, 548

usage
   Dakota::GetLongOpt, 247

var_based_decomp
   Dakota::Analyzer, 80

Variables
   Dakota::Variables, 591, 592

variables_recast
   Dakota::Minimizer, 327

vars_set_compare
   Dakota, 58

Vector
   Dakota::Vector, 598

VMAX
   Dakota::NonDEvidence, 410

VMIN
   Dakota::NonDEvidence, 410

volumetric_quality
   Dakota::Analyzer, 80

write
   Dakota::ParamResponsePair, 484

write_annotated
   Dakota::ResponseRep, 517

write_tabular
   Dakota::ResponseRep, 517
X
  Dakota::NonDEvidence, 411

Y
  Dakota::NonDEvidence, 410