SAND2006-4056
Unlimited Release
October 2006

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

Version 4.0 Developers Manual

Michael S. Eldred, Shannon L. Brown, Brian M. Adams, Daniel M. Dunlavy, David M. Gay, Laura P. Swiler
Optimization and Uncertainty Estimation Department

Anthony A. Giunta
Validation and Uncertainty Quantification Processes Department

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

John P. Eddy
System Sustainment and Readiness Technologies Department

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

Josh D. Griffin, Patty D. Hough, Tammy G. Kolda, Monica L. Martinez-Canales, Pamela J. Williams
Computational Sciences and Mathematics Research 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

Chapter 1

DAKOTA Developers Manual

Author:

Michael S. Eldred, Anthony A. Giunta, Shannon L. Brown, Brian M. Adams, Daniel M. Dunlavy, 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 and JEGAOptimizer nongradient-based optimization 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), and NonDSampling. NonDSampling is further specialized with the NonDLHSSampling class for latin hypercube and Monte Carlo sampling and the NonDPCESampling class for polynomial chaos expansions.
- Parameter studies and design of experiments: PStudyDACE provides a base class for ParamStudy, which provides capabilities for directed parameter space interrogation, 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.

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.
- 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, HermiteApproximation, 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 sharing through the use of letter-envelope and handle-body idioms (Coplien, "Advanced C++"). The former idiom provides for memory efficiency and enhanced polymorphism in the following class hierarchies: Strategy, Iterator, Model, Variables, Constraints, Interface, ProblemDescDB, and Approximation. The latter idiom provides for memory efficiency in data-intensive classes which do not involve a class hierarchy. 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

- 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.

- Project web pages are maintained at `http://endo.sandia.gov/DAKOTA` with software specifics and documentation pointers provided at `http://endo.sandia.gov/DAKOTA/software.html`, and a list of publications provided at `http://endo.sandia.gov/DAKOTA/references.html`

Chapter 2

DAKOTA Directory Hierarchy

2.1 DAKOTA Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 3

DAKOTA Namespace Index

3.1 DAKOTA Namespace List

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

Chapter 4

DAKOTA Hierarchical Index

4.1 DAKOTA Class Hierarchy

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

Chapter 5

DAKOTA Class Index

5.1 DAKOTA Class List

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

Chapter 6

DAKOTA File Index

6.1 DAKOTA File List

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

Chapter 7

DAKOTA Page Index

7.1 DAKOTA Related Pages

Here is a list of all related documentation pages:

Chapter 8

DAKOTA Directory Documentation

8.1 /home/mseldre/dev/Dakota/src/ Directory Reference

Files

- file **AllConstraints.C**
- file **AllConstraints.H**
- file **AllVariables.C**
- file **AllVariables.H**
- file **AnalysisCode.C**
- file **AnalysisCode.H**
- file **ApplicationInterface.C**
- file **ApplicationInterface.H**
- file **ApproximationInterface.C**
- file **ApproximationInterface.H**
- file **COLINApplication.C**
- file **COLINApplication.H**
- file **COLINOptimizer.H**
- file **CommandLineHandler.C**
- file **CommandLineHandler.H**
- file **CommandShell.C**
- file **CommandShell.H**
- file **ConcurrentStrategy.C**
- file **ConcurrentStrategy.H**
- file **CONMINOptimizer.C**
- file **CONMINOptimizer.H**
- file **CtelRegExp.C**
- file **CtelRegExp.H**
- file **DakotaActiveSet.C**

- file **DakotaActiveSet.H**
- file **DakotaAnalyzer.C**
- file **DakotaAnalyzer.H**
- file **DakotaApproximation.C**
- file **DakotaApproximation.H**
- file **DakotaArray.H**
- file **DakotaBaseVector.H**
- file **DakotaBinStream.C**
- file **DakotaBinStream.H**
- file **DakotaConstraints.C**
- file **DakotaConstraints.H**
- file **DakotaGraphics.C**
- file **DakotaGraphics.H**
- file **DakotaInterface.C**
- file **DakotaInterface.H**
- file **DakotaIterator.C**
- file **DakotaIterator.H**
- file **DakotaLeastSq.C**
- file **DakotaLeastSq.H**
- file **DakotaList.H**
- file **DakotaMatrix.H**
- file **DakotaMinimizer.C**
- file **DakotaMinimizer.H**
- file **DakotaModel.C**
- file **DakotaModel.H**
- file **DakotaNonD.C**
- file **DakotaNonD.H**
- file **DakotaOptimizer.C**
- file **DakotaOptimizer.H**
- file **DakotaPStudyDACE.C**
- file **DakotaPStudyDACE.H**
- file **DakotaResponse.C**
- file **DakotaResponse.H**
- file **DakotaStrategy.C**
- file **DakotaStrategy.H**
- file **DakotaString.C**
- file **DakotaString.H**
- file **DakotaVariables.C**
- file **DakotaVariables.H**
- file **DakotaVector.H**
- file **data_types.C**
- file **data_types.h**
- file **DataFitSurrModel.C**
- file **DataFitSurrModel.H**
- file **DataInterface.C**
- file **DataInterface.H**

- file **DataMethod.C**
- file **DataMethod.H**
- file **DataModel.C**
- file **DataModel.H**
- file **DataResponses.C**
- file **DataResponses.H**
- file **DataStrategy.C**
- file **DataStrategy.H**
- file **DataVariables.C**
- file **DataVariables.H**
- file **DDACEDesignCompExp.C**
- file **DDACEDesignCompExp.H**
- file **DirectFnApplicInterface.C**
- file **DirectFnApplicInterface.H**
- file **DistinctConstraints.C**
- file **DistinctConstraints.H**
- file **DistinctVariables.C**
- file **DistinctVariables.H**
- file **DOTOptimizer.C**
- file **DOTOptimizer.H**
- file **ForkAnalysisCode.C**
- file **ForkAnalysisCode.H**
- file **ForkApplicInterface.C**
- file **ForkApplicInterface.H**
- file **FSUDesignCompExp.C**
- file **FSUDesignCompExp.H**
- file **GaussProcApproximation.C**
- file **GaussProcApproximation.H**
- file **global_defs.C**
- file **global_defs.h**
- file **GridApplicInterface.C**
- file **GridApplicInterface.H**
- file **HermiteApproximation.C**
- file **HermiteApproximation.H**
- file **HermiteChaos.C**
- file **HermiteChaos.H**
- file **HierarchSurrModel.C**
- file **HierarchSurrModel.H**
- file **IDRProblemDescDB.C**
- file **IDRProblemDescDB.H**
- file JEGAEvaluator.C

  Contains the implementation of the JEGAEvaluator class.

- file JEGAEvaluator.H

  Contains the definition of the JEGAEvaluator class.

- file JEGAOptimizer.C

  Contains the implementation of the JEGAOptimizer class.

- file JEGAOptimizer.H

  Contains the definition of the JEGAOptimizer class.

- file keywordtable.C

  file containing keywords for the strategy, method, model, variables, interface, and responses input specifications from **dakota.input.spec**

- file **LatinHypercube.C**
- file **LatinHypercube.H**
- file **LHSInput.C**
- file **LHSInput.H**
- file main.C

  file containing the main program for DAKOTA

- file **MergedConstraints.C**
- file **MergedConstraints.H**
- file **MergedVariables.C**
- file **MergedVariables.H**
- file **MPIPackBuffer.C**
- file **MPIPackBuffer.H**
- file **MultilevelOptStrategy.C**
- file **MultilevelOptStrategy.H**
- file **NestedModel.C**
- file **NestedModel.H**
- file **NL2SOLLeastSq.C**
- file **NL2SOLLeastSq.H**
- file **NLPQLPOptimizer.C**
- file **NLPQLPOptimizer.H**
- file **NLSSOLLeastSq.C**
- file **NLSSOLLeastSq.H**
- file **NonDEvidence.C**
- file **NonDEvidence.H**
- file **NonDLHSSampling.C**
- file **NonDLHSSampling.H**
- file **NonDPCESampling.C**
- file **NonDPCESampling.H**
- file **NonDReliability.C**
- file **NonDReliability.H**
- file **NonDSampling.C**
- file **NonDSampling.H**
- file **NPSOLOptimizer.C**
- file **NPSOLOptimizer.H**
- file **ParallelLibrary.C**

- file **ParallelLibrary.H**
- file **ParamResponsePair.C**
- file **ParamResponsePair.H**
- file **ParamStudy.C**
- file **ParamStudy.H**
- file **PluginDirectFnApplicInterface.C**
- file **PluginDirectFnApplicInterface.H**
- file **ProblemDescDB.C**
- file **ProblemDescDB.H**
- file **regexp.h**
- file restart_util.C

    file containing the DAKOTA restart utility main program

- file **SingleMethodStrategy.C**
- file **SingleMethodStrategy.H**
- file **SingleModel.C**
- file **SingleModel.H**
- file **SNLLBase.C**
- file **SNLLBase.H**
- file **SNLLLeastSq.C**
- file **SNLLLeastSq.H**
- file **SNLLOptimizer.C**
- file **SNLLOptimizer.H**
- file **SOLBase.C**
- file **SOLBase.H**
- file **SurfpackApproximation.C**
- file **SurfpackApproximation.H**
- file **SurrBasedOptStrategy.C**
- file **SurrBasedOptStrategy.H**
- file **SurrogateModel.C**
- file **SurrogateModel.H**
- file **SysCallAnalysisCode.C**
- file **SysCallAnalysisCode.H**
- file **SysCallApplicInterface.C**
- file **SysCallApplicInterface.H**
- file **system_defs.h**
- file **TANA3Approximation.C**
- file **TANA3Approximation.H**
- file **TaylorApproximation.C**
- file **TaylorApproximation.H**
- file **template_defs.h**
- file **VariablesUtil.H**

Chapter 9

DAKOTA Namespace Documentation

9.1 Dakota Namespace Reference

The primary namespace for DAKOTA.

Classes

- class AllConstraints

  Derived class within the Constraints hierarchy which employs the all data view.

- class AllVariables

  Derived class within the Variables hierarchy which employs the all data view.

- class AnalysisCode

  Base class providing common functionality for derived classes (SysCallAnalysisCode and ForkAnalysisCode) which spawn separate processes for managing simulations.

- class ApplicationInterface

  Derived class within the interface class hierarchy for supporting interfaces to simulation codes.

- class ApproximationInterface

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

- class COLINApplication
- class COLINOptimizer

  Wrapper class for optimizers defined using COLIN.

- class GetLongOpt

GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

- class CommandLineHandler

  Utility class for managing command line inputs to DAKOTA.

- class CommandShell

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

- class ConcurrentStrategy

  Strategy for multi-start iteration or pareto set optimization.

- class CONMINOptimizer

  Wrapper class for the CONMIN optimization library.

- class ActiveSet

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

- class Analyzer

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

- class SurrogateDataPoint

  Container class encapsulating basic parameter and response data for defining a "truth" data point.

- class SurrogateDataPointRep

  The representation of a surrogate data point. This representation, 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

  The binary input stream class. Overloads the >> operator for all data types.

- class BoStream

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

- class Constraints

  Base class for the variable constraints class hierarchy.

- class Graphics

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

- class Interface

  Base class for the interface class hierarchy.

- class Iterator

  Base class for the iterator class hierarchy.

- class LeastSq

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

- class List

  Template class for the Dakota bookkeeping list.

- class FunctionCompare
- class SortCompare
- class Matrix

  Template class for the Dakota numerical matrix.

- class Minimizer

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

- class Model

  Base class for the model class hierarchy.

- class NonD

  Base class for all nondetermistic iterators (the DAKOTA/UQ branch).

- class Optimizer

  Base class for the optimizer branch of the iterator hierarchy.

- class PStudyDACE

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

- class Response

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

- class ResponseRep

  Container class for response functions and their derivatives. 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

  Derived model class within the surrogate model branch for managing 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

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

- class DistinctConstraints

  Derived class within the Constraints hierarchy which employs the default data view (no variable or domain type array merging).

- class DistinctVariables

  Derived class within the Variables hierarchy which employs the default data view (no variable or domain type array merging).

- class DOTOptimizer

  Wrapper class for the DOT optimization library.

- class ForkAnalysisCode

  Derived class in the AnalysisCode class hierarchy which spawns simulations using forks.

- class ForkApplicInterface

  Derived application interface class which spawns simulation codes using forks.

- class FSUDesignCompExp

  Wrapper class for the FSUDace QMC/CVT library.

- class GaussProcApproximation

  Derived approximation class for Gaussian Process implementation.

- struct BaseConstructor

  Dummy struct for overloading letter-envelope constructors.

- struct NoDBBaseConstructor

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

- class GridApplicInterface

  Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.

- class HermiteApproximation

  Derived approximation class for Hermite polynomials (global approximation).

- class HierarchSurrModel

  Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

- class IDRProblemDescDB

  The derived input file database utilizing the IDR parser.

- class JEGAEvaluator

  This evaluator uses Sandia National Laboratories Dakota software.

- class JEGAOptimizer

  Version of Optimizer for instantiation of John Eddy's Genetic Algorithms.

- class MergedConstraints

  Derived class within the Constraints hierarchy which employs the merged data view.

- class MergedVariables

  Derived class within the Variables hierarchy which employs the merged data view.

- class MPIPackBuffer

  Class for packing MPI message buffers.

- class MPIUnpackBuffer

  Class for unpacking MPI message buffers.

- class MultilevelOptStrategy

  Strategy for hybrid optimization using multiple optimizers on multiple models of varying fidelity.

- class NestedModel

  Derived model class which performs a complete sub-iterator 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 NonDEvidence

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

- class NonDLHSSampling

  Performs LHS and Monte Carlo sampling for uncertainty quantification.

- class NonDPCESampling

  Stochastic finite element approach to uncertainty quantification using polynomial chaos expansions.

- class NonDReliability

  Class for the reliability methods within DAKOTA/UQ.

- class NonDSampling

  Base class for common code between NonDLHSSampling and NonDPCESampling.

- class NPSOLOptimizer

  Wrapper class for the NPSOL optimization library.

- class ParallelLevel

  Container class for the data associated with a single level of communicator partitioning.

- class ParallelConfiguration

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

- class ParallelLibrary

  Class for partitioning multiple levels of parallelism and managing message passing within these levels.

- class ParamResponsePair

  Container class for a variables object, a response object, and an evaluation id.

- class ParamStudy

  Class for vector, list, centered, and multidimensional parameter studies.

- class ProblemDescDB

  The database containing information parsed from the DAKOTA input file.

- class SingleMethodStrategy

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

- class SingleModel

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

- class SNLLBase

  Base class for OPT++ optimization and least squares methods.

- class SNLLLeastSq

  Wrapper class for the OPT++ optimization library.

- class SNLLOptimizer

  Wrapper class for the OPT++ optimization library.

- class SOLBase

  Base class for Stanford SOL software.

- class SurfpackApproximation

  Derived approximation class for Surfpack approximation classes. 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

  Derived class in the AnalysisCode class hierarchy which spawns simulations using system calls.

- class SysCallApplicInterface

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

- class TANA3Approximation

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

- class TaylorApproximation

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

- class VariablesUtil

  Utility class for the Variables and Constraints hierarchies which provides convenience functions for variable vectors and label arrays for combining design, uncertain, and state variable types and merging continuous and discrete variable domains.

Typedefs

- typedef Vector< Real > **RealVector**
- typedef Vector< int > **IntVector**
- typedef BaseVector< Real > **RealBaseVector**
- typedef Matrix< Real > **RealMatrix**
- typedef Matrix< int > **IntMatrix**
- typedef Array< Real > **RealArray**
- typedef Array< int > **IntArray**
- typedef Array< size_t > **SizetArray**
- typedef Array< String > **StringArray**
- typedef Array< StringArray > **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::map< int, short > **IntShortMap**

- typedef std::map< int, RealVector > **IntRealVectorMap**
- typedef std::map< int, Response > **IntResponseMap**
- typedef IntList::iterator **ILIter**
- typedef IntList::const_iterator **ILCIter**
- typedef SizetList::iterator **StLIter**
- typedef SizetList::const_iterator **StLCIter**
- typedef RealList::iterator **RLIter**
- typedef RealList::const_iterator **RLCIter**
- typedef StringList::iterator **StringLIter**
- typedef StringList::const_iterator **StringLCIter**
- typedef RealVectorList::iterator **RVLIter**
- typedef RealVectorList::const_iterator **RVLCIter**
- typedef VariablesList::iterator **VarsLIter**
- typedef InterfaceList::iterator **InterfLIter**
- typedef ResponseList::iterator **RespLIter**
- typedef ModelList::iterator **ModelLIter**
- typedef IteratorList::iterator **IterLIter**
- typedef PRPList::iterator **PRPLIter**
- typedef List< ParallelLevel >::iterator **ParLevLIter**
- typedef List< ParallelConfiguration >::iterator **ParConfigLIter**
- typedef IntSet::iterator **ISIter**
- typedef IntShortMap::iterator **IntShMIter**
- typedef IntRealVectorMap::iterator **IntRVMIter**
- typedef IntResponseMap::iterator **IntRespMIter**
- typedef IntResponseMap::const_iterator **IntRespMCIter**
- typedef int(∗ **start_grid_computing_t**)(char ∗analysis_driver_script, char ∗params_file, char ∗results_-file)
- typedef int(∗ **perform_analysis_t**)(char ∗iteration_num)
- typedef int ∗(∗ **get_jobs_completed_t**)()
- typedef int(∗ **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(∗ **Calcrj**)(int ∗n, int ∗p, Real ∗x, int ∗nf, Real ∗r, int ∗ui, void ∗ur, Vf vf)
- typedef void(∗ **Vf**)()

Enumerations

- enum **LHSNames** {
  NORMAL, **LOGNORMAL**, **UNIFORM**, **LOGUNIFORM**,
  WEIBULL, **CONSTANT**, **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**,

  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 {

  NORMAL, **LOGNORMAL**, **UNIFORM**, **LOGUNIFORM**,

  TRIANGULAR, **BETA**, **GAMMA**, **GUMBEL**,

  FRECHET, **WEIBULL** }
- enum {

  MV, **AMV_X**, **AMV_U**, **AMV_PLUS_X**,

  AMV_PLUS_U, **TANA_X**, **TANA_U**, **NO_APPROX** }
- enum **EvalType** { **NLFEvaluator**, **CONEvaluator** }
- enum { **NONE** = 0, **HOMOTOPY** = 1, **COMPOSITE_STEP** = 2 }
- enum { **BASIC_PENALTY**, **ADAPTIVE_PENALTY**, **BASIC_LAGRANGIAN**, **AUGMENTED_-LAGRANGIAN** }
- enum { **FILTER**, **TR_RATIO** }
- 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

- template<> void COLINOptimizer< coliny::DIRECT >::set_method_parameters (void)

  ————————————————— Section 3———————————————————

- template<> void COLINOptimizer< coliny::Cobyla >::set_method_parameters (void)
- template<> void COLINOptimizer< coliny::APPS >::set_method_parameters (void)
- template<> void COLINOptimizer< coliny::PatternSearch >::set_runtime_parameters ()
- template<> void COLINOptimizer< coliny::PatternSearch >::set_method_parameters (void)
- template<> void COLINOptimizer< coliny::SolisWets >::set_method_parameters (void)
- template<> void COLINOptimizer< coliny::EAminlp >::set_method_parameters (void)
- 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(istream&).

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

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

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

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

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

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

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

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

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

  inequality operator

- 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(istream&).

- BiStream & operator>> (BiStream &s, Response &response)

  BiStream extraction operator for Response. Calls read(BiStream&).

- BoStream & operator<< (BoStream &s, const Response &response)

  BoStream insertion operator for Response. Calls write(BoStream&).

- MPIUnpackBuffer & operator>> (MPIUnpackBuffer &s, Response &response)

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

- MPIPackBuffer & operator<< (MPIPackBuffer &s, const Response &response)

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

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

  equality operator

- 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 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)
  copy RealMatrix to NEWMAT::SymmetricMatrix
- void copy_data (const RealMatrix &drm, NEWMAT::Matrix &m)
  copy RealMatrix to NEWMAT::Matrix
- void copy_data (const Epetra_SerialDenseVector &psdv, RealVector &drv)
  copy Epetra_SerialDenseVector to RealVector
- void copy_data (const Epetra_SerialDenseVector &psdv, RealBaseVector &drbv)
  copy Epetra_SerialDenseVector to RealBaseVector
- void copy_data (const Epetra_SerialDenseMatrix &psdm, RealMatrix &drm)
  copy Epetra_SerialDenseMatrix to RealMatrix
- void copy_data (const Epetra_SerialSymDenseMatrix &pssdm, RealMatrix &drm)
  copy Epetra_SerialSymDenseMatrix to RealMatrix
- void copy_data (const RealVector &drv, Epetra_SerialDenseVector &psdv)
  copy RealVector to Epetra_SerialDenseVector
- void copy_data (const RealArray &dra, Epetra_SerialDenseVector &psdv)
  copy RealArray to Epetra_SerialDenseVector
- void copy_data (const RealBaseVector &drbv, Epetra_SerialDenseVector &psdv)
  copy RealBaseVector to Epetra_SerialDenseVector

- void copy_data (const Real ∗ptr, const int ptr_len, Epetra_SerialDenseVector &psdv)

  copy Real∗ to Epetra_SerialDenseVector

- void copy_data (const RealMatrix &drm, Epetra_SerialDenseMatrix &psdm)

  copy RealMatrix to Epetra_SerialDenseMatrix

- void copy_data (const RealMatrix &drm, Epetra_SerialSymDenseMatrix &pssdm)

  copy RealMatrix to Epetra_SerialSymDenseMatrix

- void copy_data (const RealMatrixArray &drma, Array< Epetra_SerialSymDenseMatrix > &pssdma)

  copy RealMatrixArray to Array<Epetra_SerialSymDenseMatrix>

- void copy_data (const NEWMAT::ColumnVector &cv, Epetra_SerialDenseVector &psdv)

  copy NEWMAT::ColumnVector to Epetra_SerialDenseVector

- void copy_data (const std::vector< DDaceSamplePoint > &dspa, RealVectorArray &drva)

  copy DDACE Array to RealVectorArray

- void copy_data (const std::vector< DDaceSamplePoint > &dspa, Real ∗ptr, const int ptr_len)

  copy DDACE Array to RealVectorArray

- 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 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)

  create an array of labels by tagging root_label for each entry in label_array. Uses build_label().

- void build_labels_partial (StringArray &label_array, const String &root_label, size_t start_index, size_t num_items)

  create a partial array of labels by tagging root_label for a subset of entries in label_array. Uses build_label().

- template<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 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 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 Vector< T > &dv, Array< Vector< T > > &dva, size_t num_-vec, size_t vec_len)

  copy Vector<T> to Array<Vector<T> >

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

  copy Array<T> to Vector<T>

- template<class T> void copy_data (const BaseVector< T > &dbv, Vector< T > &dv)

  copy BaseVector<T> to Vector<T>

- template<class T> void copy_data (const List< T > &dl, Array< T > &da)

  copy List<T> to Array<T>

- template<class T> void copy_data (const List< T > &dl, Array< Array< T > > &d2a, size_t num_a, size_t a_len)

  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 (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

- int salinas_main (int argc, char ∗argv[ ], MPI_Comm ∗comm)

  subroutine interface to SALINAS simulation code

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

  equality operator

- ParallelLibrary dummy_lib (0)

  dummy ParallelLibrary object used for mandatory reference initialization when a real ParallelLibrary instance is unavailable

- ProblemDescDB dummy_db (dummy_lib)

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

- void abort_handler (int code)

  global function which handles serial or parallel aborts

- int **start_grid_computing** (char ∗analysis_driver_script, char ∗params_file, char ∗results_file)
- int **stop_grid_computing** ()
- int **perform_analysis** (char ∗iteration_num)
- template<typename T> string asstring (const T &val)

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

- 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

- 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 ∗liv, int ∗lv, Real ∗v, int ∗ui, void ∗ur, Vf)
- void **dn2fb_** (int ∗n, int ∗p, Real ∗x, Real ∗b, Calcrj, int ∗iv, int ∗liv, int ∗lv, Real ∗v, int ∗ui, void ∗ur, Vf)
- void **dn2g_** (int ∗n, int ∗p, Real ∗x, Calcrj, Calcrj, int ∗iv, int ∗liv, int ∗lv, Real ∗v, int ∗ui, void ∗ur, Vf)
- void **dn2gb_** (int ∗n, int ∗p, Real ∗x, Real ∗b, Calcrj, Calcrj, int ∗iv, int ∗liv, 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 &s, ParamResponsePair &pair)

  istream extraction operator for ParamResponsePair

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

  ostream insertion operator for ParamResponsePair

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

  BiStream extraction operator for ParamResponsePair.

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

  BoStream insertion operator for ParamResponsePair.

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

  MPIUnpackBuffer extraction operator for ParamResponsePair.

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

  MPIPackBuffer insertion operator for ParamResponsePair.

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

  inequality operator

- 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

- 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

- ParallelLibrary dummy_lib

  dummy ParallelLibrary object used for mandatory reference initialization when a real ParallelLibrary 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

- 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

- class class class class class class class typedef double **Real**
- 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()).

- int write_precision = 10

  used in ostream data output functions (restart_util.C overrides this default value)

- int mc_ptr_int = 0

  global pointer for ModelCenter API

- FILE ∗ **yyin**
- const int **MAXPOSDEF** = 10
- const int **NONRANDOM** = 0
- const int **RANDOM** = 1
- Dakota::GSL_Singleton **GSL_RNG**
- const int **LARGE_SCALE** = 100
- const size_t _NPOS = ∼(size_t)0

  special value returned by index() when entry not found

9.1.1 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 VendorOptimizers and VendorPackages. The C++ source files defining these core classes reside in Dakota/src as ∗.[CH].

9.1.2 Function Documentation

9.1.2.1 void COLINOptimizer< coliny::DIRECT >::set_method_parameters (void)

Section 3

specialization of set_method_parameters() for DIRECT

9.1.2.2 void COLINOptimizer< coliny::Cobyla >::set_method_parameters (void)

specialization of set_method_parameters() for Cobyla

9.1.2.3 void COLINOptimizer< coliny::APPS >::set_method_parameters (void)

specialization of set_method_parameters() for APPS

9.1.2.4 void COLINOptimizer< coliny::PatternSearch >::set_runtime_parameters ()

specialization of set_runtime_parameters() for PatternSearch

9.1.2.5 void COLINOptimizer< coliny::PatternSearch >::set_method_parameters (void)

specialization of set_method_parameters() for PatternSearch

9.1.2.6 void COLINOptimizer< coliny::SolisWets >::set_method_parameters (void)

specialization of set_method_parameters() for SolisWets

9.1.2.7 void COLINOptimizer< coliny::EAminlp >::set_method_parameters (void)

specialization of set_method_parameters() for EAminlp

9.1.2.8 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.

9.1.2.9 bool operator== (const DistinctVariables & *vars1*, const DistinctVariables & *vars2*)

equality operator

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

9.1.2.10 bool 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.

9.1.2.11 bool 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.

9.1.2.12 bool eval_id_sort_fn (const ParamResponsePair & *pr1*, const ParamResponsePair & *pr2*) `[inline]`

sort function for ParamResponsePair

a global function used to sort a PRPList by evalId's.

9.1.2.13 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.

9.1.2.14 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/TECPLOT data file).

9.1.2.15 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.

9.1.2.16 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).

9.1.2.17 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.

9.2 SIM Namespace Reference

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA.

Classes

- class DirectFnApplicInterface

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

9.2.1 Detailed Description

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA.

A typical use of plug-ins with assign_rep() is to publish a simulation interface for use in library mode See Interfacing with DAKOTA as a Library for more information.

Chapter 10

DAKOTA Class Documentation

10.1 ActiveSet Class Reference

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

Public Member Functions

- ActiveSet ()

  default constructor

- ActiveSet (size_t num_fns, size_t num_deriv_vars)

  standard constructor

- ActiveSet (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 IntArray & request_vector () const

  return the request vector

- void request_vector (const IntArray &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

- IntArray requestVector
  the vector of response requests

- IntArray derivVarsVector
  the vector of variable ids used for computing derivatives

Friends

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

10.1.1 Detailed Description

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

The ActiveSet class is a small class whose initial design function is to avoid having to pass the ASV and DVV separately. It is not part of a class hierarchy and does not employ reference-counting/ representation-sharing idioms (e.g., handle-body).

10.1.2 Member Data Documentation

10.1.2.1 IntArray 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.

10.1.2.2 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.

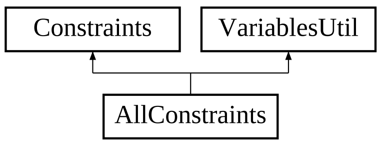
The documentation for this class was generated from the following files:

- DakotaActiveSet.H
- DakotaActiveSet.C

10.2 AllConstraints Class Reference

Derived class within the Constraints hierarchy which employs the all data view.

Inheritance diagram for 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

- RealVector all_continuous_upper_bounds () const

  returns a single array with all continuous upper bounds

- IntVector all_discrete_lower_bounds () const

  returns a single array with all discrete lower bounds

- IntVector all_discrete_upper_bounds () const

  returns a single array with all discrete upper bounds

- 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

Private Attributes

- RealVector allContinuousLowerBnds

  a continuous lower bounds array combining continuous design, uncertain, and continuous state variable types (all view).

- RealVector allContinuousUpperBnds

  a continuous upper bounds array combining continuous design, uncertain, and continuous state variable types (all view).

- IntVector allDiscreteLowerBnds

  a discrete lower bounds array combining discrete design and discrete state variable types (all view).

- IntVector allDiscreteUpperBnds

  a discrete upper bounds array combining discrete design and discrete state variable types (all view).

- size_t numCDV

  number of continuous design variables

- size_t numDDV

  number of discrete design variables

- size_t numUV

number of uncertain variables

- size_t numCSV

  number of continuous state variables

- size_t numDSV

  number of discrete state variables

10.2.1 Detailed Description

Derived class within the Constraints hierarchy which 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).

10.2.2 Constructor & Destructor Documentation

10.2.2.1 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.

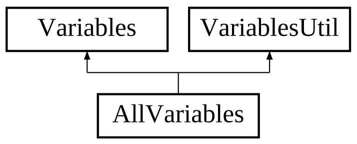
The documentation for this class was generated from the following files:

- AllConstraints.H
- AllConstraints.C

10.3 AllVariables Class Reference

Derived class within the Variables hierarchy which employs the all data view.

Inheritance diagram for 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

- StringArray all_discrete_variable_labels () const

  returns a single array with all discrete variable labels

- 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

Private Member Functions

- void copy_rep (const Variables ∗vars_rep)

  Used by copy() to copy the contents of a letter class.

Private Attributes

- RealVector allContinuousVars

  a continuous array combining all of the continuous variables (design, uncertain, and state).

- IntVector allDiscreteVars

  a discrete array combining all of the discrete variables (design and state).

- StringArray allContinuousLabels

  a label array combining all of the continuous variable labels (design, uncertain, and state).

- StringArray allDiscreteLabels

  a label array combining all of the discrete variable labels (design and state).

- size_t numCDV

  number of continuous design variables

- size_t numDDV

  number of discrete design variables

- size_t numUV

  number of uncertain variables

- size_t numCSV

  number of continuous state variables

- size_t numDSV

  number of discrete state variables

Friends

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

  equality operator

10.3.1 Detailed Description

Derived class within the Variables hierarchy which employs 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)).

10.3.2 Constructor & Destructor Documentation

10.3.2.1 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.

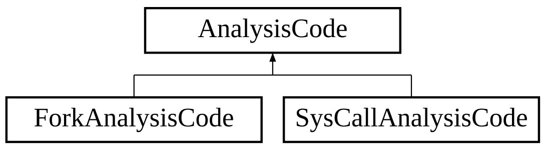
The documentation for this class was generated from the following files:

- AllVariables.H
- AllVariables.C

10.4 AnalysisCode Class Reference

Base class providing common functionality for derived classes (SysCallAnalysisCode and ForkAnalysisCode) which spawn separate processes for managing simulations.

Inheritance diagram for AnalysisCode::



Public Member Functions

- void define_filenames (const int id)

  define modified filenames from user input by handling Unix temp file and tagging options

- void write_parameters_files (const Variables &vars, const ActiveSet &set, const int id)

  write the parameters data and response request data to one or more parameters files (using one or more invocations of write_parameters_file()) in either standard or aprepro format

- void read_results_files (Response &response, const int id)

  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

- bool suppress_output_flag () const

  return suppressOutputFlag

- bool multiple_parameters_filenames () const

  return multipleParamsFiles

Protected Member Functions

- AnalysisCode (const ProblemDescDB &problem_db)

  constructor

- virtual ∼AnalysisCode ()

  destructor

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 apreproFlag

  flags use of the APREPRO (the Sandia "A PRE PROcessor" utility) format for parameter files

- bool multipleParamsFiles

  flag indicating the need for separate parameters files for multiple analysis drivers

- 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

  the names of the analysis code programs (analysis_drivers user 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

  the parameters file name actually used (modified with tagging or 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

  stores parameters and results file names used in spawning function 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)

  write the variables, active set vector, derivative variables vector, and analysis components to the specified parameters file in either standard or aprepro format

Private Attributes

- ParallelLibrary & parallelLib

  reference to the ParallelLibrary object. Used in define_filenames().

- String2DArray analysisComponents

  the set of optional analysis components used by the analysis drivers (from the analysis_components interface specification)

10.4.1 Detailed Description

Base class providing common functionality for derived classes (SysCallAnalysisCode and ForkAnalysisCode) which spawn separate 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.

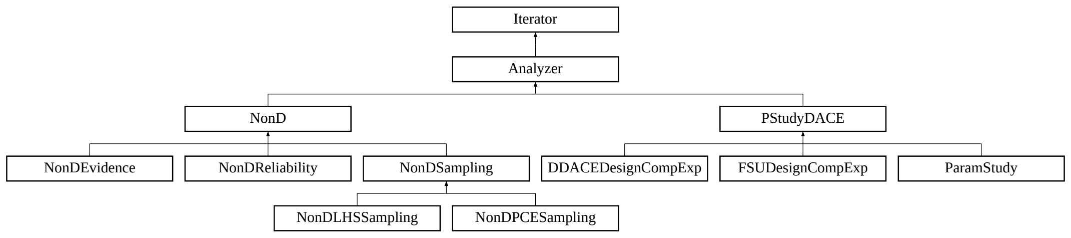
The documentation for this class was generated from the following files:

- AnalysisCode.H
- AnalysisCode.C

10.5 Analyzer Class Reference

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

Inheritance diagram for Analyzer::



Public Member Functions

- const VariablesArray & all_variables () const

  return the complete set of evaluated variables

- const RealVectorArray & all_c_variables () const

  return the complete set of evaluated continuous variables

- const ResponseArray & all_responses () const

  return the complete set of computed responses

- const RealVectorArray & all_fn_responses () const

  return the complete set of computed function responses

Protected Member Functions

- Analyzer ()

  default constructor

- Analyzer (Model &model)

  standard constructor

- Analyzer (NoDBBaseConstructor, Model &model)

  alternate constructor for instantiations "on the fly"

- ∼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 ()

  Returns one block of samples (ndim ∗ num_samples).

- void evaluate_parameter_sets (bool vars_flag, bool resp_flag, bool fns_flag, bool best_flag)

  perform function evaluations to map parameter sets (allVariables/allCVariables/allDVariables) into response sets (allResponses/allFnResponses/allGradResponses)

- 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 RealVector &S, const RealVector &T) const

  Printing of VBD results.

Protected Attributes

- VariablesArray allVariables

  array of all variables evaluated

- RealVectorArray allCVariables

  array of all continuous variables evaluated (subset of allVariables)

- ResponseArray allResponses

  array of all responses computed

- RealVectorArray allFnResponses

  array of all function responses computed (subset of allResponses)

- StringArray allHeaders

  array of headers to insert into output while evaluating allCVariables

- bool qualityFlag

  flag to indicated if quality metrics were calculated

- double chiMeas

  quality measures

- double dMeas

quality measures

- double hMeas

  quality measures

- double tauMeas

  quality measures

10.5.1 Detailed Description

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

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

10.5.2 Member Function Documentation

10.5.2.1 void evaluate_parameter_sets (bool *vars_flag*, bool *resp_flag*, bool *fns_flag*, bool *best_flag*) `[protected]`

perform function evaluations to map parameter sets (allVariables/allCVariables/allDVariables) into response sets (allResponses/allFnResponses/allGradResponses)

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

10.5.2.2 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.

10.5.2.3 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.

10.5.2.4 void print_vbd (ostream & *s*, const RealVector & *S*, const RealVector & *T*) const `[protected]`

Printing of VBD results.

printing of variance based decomposition indices.

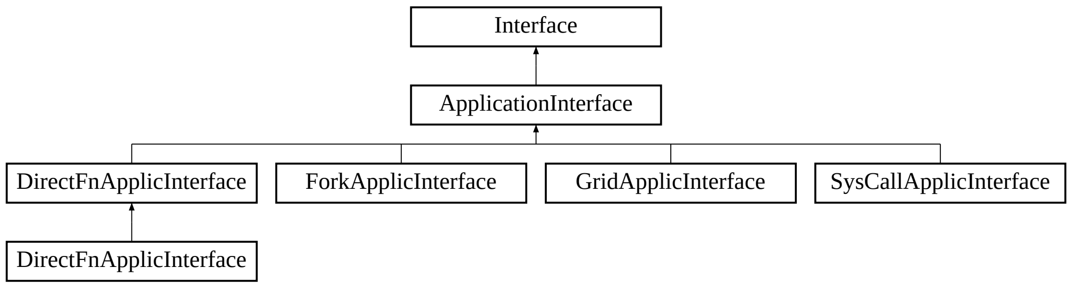
The documentation for this class was generated from the following files:

- DakotaAnalyzer.H
- DakotaAnalyzer.C

10.6 ApplicationInterface Class Reference

Derived class within the interface class hierarchy for supporting interfaces to simulation codes.

Inheritance diagram for ApplicationInterface::



Public Member Functions

- ApplicationInterface (const ProblemDescDB &problem_db)

  constructor

- ∼ApplicationInterface ()

  destructor

Protected Member Functions

- void init_communicators (const IntArray &message_lengths, const int &max_iterator_concurrency)

  allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- void reset_communicators (const IntArray &message_lengths)

  reset the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

- void free_communicators ()

  deallocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- void init_serial ()
- int asynch_local_evaluation_concurrency () const

  return asynchLocalEvalConcurrency

- String interface_synchronization () const *return interfaceSynchronization*

- void map (const Variables &vars, const ActiveSet &set, Response &response, const bool asynch_-flag=false)

  Provides a "mapping" of variables to responses using a simulation. Protected due to Interface letter-envelope idiom.

- void manage_failure (const Variables &vars, const ActiveSet &set, Response &response, int failed_eval_-id)

  manages a simulation failure using abort/retry/recover/continuation

- const ResponseArray & synch ()

  executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPList queue and returns all jobs

- const IntResponseMap & synch_nowait ()

  executes a nonblocking schedule for asynchronous evaluations in the 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)

  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.

- virtual void derived_map_asynch (const ParamResponsePair &pair)

  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.

- virtual void derived_synch (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- virtual void derived_synch_nowait (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- void self_schedule_analyses ()

  blocking self-schedule of all analyses within a function evaluation using message passing

- void serve_analyses_synch ()

  serve the master analysis scheduler and manage one synchronous analysis job at a time

- virtual int derived_synchronous_local_analysis (const int &analysis_id)

  Execute a particular analysis (identified by analysis_id) synchronously on the local processor. Used for the derived class specifics within ApplicationInterface::serve_analyses_synch().

Protected Attributes

- ParallelLibrary & parallelLib

  reference to the ParallelLibrary object used to manage MPI partitions for the concurrent evaluations and concurrent analyses parallelism levels

- bool suppressOutput

  flag for suppressing output on slave processors

- int evalCommSize

  size of evalComm

- int evalCommRank

  processor rank within evalComm

- int evalServerId

  evaluation server identifier

- bool eaDedMasterFlag

  flag for dedicated master partitioning at ea level

- int analysisCommSize

  size of analysisComm

- int analysisCommRank

  processor rank within analysisComm

- int analysisServerId

  analysis server identifier

- int numAnalysisServers

  number of analysis servers

- bool multiProcAnalysisFlag

  flag for multiprocessor analysis partitions

- bool asynchLocalAnalysisFlag

  flag for asynchronous local parallelism of analyses

- int asynchLocalAnalysisConcurrency

limits the number of concurrent analyses in asynchronous local scheduling and specifies hybrid concurrency when message passing

- int numAnalysisDrivers

  the number of analysis drivers used for each function evaluation (from the analysis_drivers interface specification)

- IntSet completionSet

  the set of completed fn_eval_id's populated by derived_synch() and derived_synch_nowait()

Private Member Functions

- bool duplication_detect (const Variables &vars, Response &response, const bool asynch_flag)

  checks data_pairs and beforeSynchCorePRPList to see if the current evaluation request has already been performed or queued

- void self_schedule_evaluations ()

  blocking self-schedule of all evaluations in beforeSynchCorePRPList using message passing; executes on iteratorComm master

- void static_schedule_evaluations ()

  blocking static schedule of all evaluations in beforeSynchCorePRPList using message passing; executes on iteratorComm master

- void asynchronous_local_evaluations (PRPList &prp_list)

  perform all jobs in prp_list using asynchronous approaches on the local processor

- void synchronous_local_evaluations (PRPList &prp_list)

  perform all jobs in prp_list using synchronous approaches on the local processor

- void asynchronous_local_evaluations_nowait (PRPList &prp_list)

  launch new jobs in prp_list asynchronously (if capacity is available), perform nonblocking query of all running jobs, and process any completed jobs

- void serve_evaluations_synch ()

  serve the evaluation message passing schedulers and perform one synchronous evaluation at a time

- void serve_evaluations_asynch ()

  serve the evaluation message passing schedulers and manage multiple asynchronous evaluations

- void serve_evaluations_peer ()

  serve the evaluation message passing schedulers and perform one synchronous evaluation at a time as part of the 1st peer

- void reset_evaluation_communicators (const IntArray &message_lengths)

  convenience function for updating the local evaluation partition data following ParallelLibrary::init_evaluation_communicators().

- void reset_analysis_communicators ()

  convenience function for updating the local analysis partition data following ParallelLibrary::init_analysis_communicators().

- const ParamResponsePair & get_source_pair (const Variables &target_vars)

  convenience function for the continuation approach in manage_failure() for finding the nearest successful "source" evaluation to the failed "target"

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

  performs a 0th order continuation method to step from a successful "source" evaluation to the failed "target". Invoked by manage_failure() for failAction == "continuation".

- void common_input_filtering (const Variables &vars)

  common input filtering operations, e.g. mesh movement with DDRIV

- 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

  length of a MPIPackBuffer containing a Variables object; computed in Model::init_communicators()

- int lenVarsActSetMessage

  length of a MPIPackBuffer containing a Variables object and an ActiveSet object; computed in Model::init_communicators()

- int lenResponseMessage

  length of a MPIPackBuffer containing a Response object; computed in Model::init_communicators()

- int lenPRPairMessage

  length of a MPIPackBuffer containing a ParamResponsePair object; computed in Model::init_communicators()

- String evalScheduling

  user specification of evaluation scheduling algorithm (self, static, or no spec). Used for manual overrides of the auto-configure logic in ParallelLibrary::resolve_inputs().

- String analysisScheduling

  user specification of analysis scheduling algorithm (self, static, or no spec). Used for manual overrides of the auto-configure logic in ParallelLibrary::resolve_inputs().

- int asynchLocalEvalConcurrency

  limits the number of concurrent evaluations in asynchronous local scheduling and specifies hybrid concurrency when message passing

- String interfaceSynchronization

  interface synchronization specification: synchronous (default) or asynchronous

- bool headerFlag

  used by synch_nowait to manage output frequency (since this function may be called many times prior to any completions)

- bool asvControlFlag

  used to manage a user request to deactivate the active set vector control. true = modify the ASV each evaluation as appropriate (default); false = ASV values are static so that the user need not check them on each evaluation.

- bool evalCacheFlag

  used to manage a user request to deactivate the function evaluation cache (i.e., queries and insertions using the data_pairs list).

- bool restartFileFlag

  used to manage a user request to deactivate the restart file (i.e., insertions into write_restart).

- IntArray defaultASV

  the static ASV values used when the user has selected asvControl = off

- String failAction

  mitigation action for captured simulation failures: abort, retry, recover, or continuation

- int failRetryLimit

  limit on the number of retries for the retry failAction

- RealVector failRecoveryFnVals

  the dummy function values used for the recover failAction

- IntList beforeSynchIdList

  bookkeeps fnEvalId's of _all_ asynchronous evaluations (new & duplicate)

- IntResponseMap historyDuplicateMap

  used to bookkeep asynchronous evaluations which duplicate data_pairs evaluations. Map key is fnEvalId, mad data is corresponding response.

- std::map< int, pair< size_t, Response > > beforeSynchDuplicateMap

  used to bookkeep fnEvalId, beforeSynchCorePRPList index, and response of asynchronous evaluations which duplicate queued beforeSynchCorePRPList evaluations

- PRPList beforeSynchCorePRPList

  used to bookkeep vars/set/response of nonduplicate asynchronous core evaluations. This is the queue of jobs populated by asynchronous map() that is later scheduled in synch() or synch_nowait().

- PRPList beforeSynchAlgPRPList

  used to bookkeep vars/set/response of asynchronous algebraic evaluations. This is the queue of algebraic jobs populated by asynchronous map() that is later evaluated in synch() or synch_nowait().

- ResponseList beforeSynchTotalRespList

  used to bookkeep total response of asynchronous evaluations with algebraic components but no core mapping components. This is populated by asynchronous map() and later used in synch() or synch_nowait().

- IntSet runningSet

  used by asynchronous_local_nowait to bookkeep which jobs are running

10.6.1 Detailed Description

Derived class within the interface class hierarchy for supporting interfaces to simulation codes.

ApplicationInterface provides an interface class for performing parameter to response mappings using simulation code(s). It provides common functionality for a number of derived classes and contains the majority of all of the scheduling algorithms in DAKOTA. The derived classes provide the specifics for managing code invocations using system calls, forks, direct procedure calls, or distributed resource facilities.

10.6.2 Member Function Documentation

10.6.2.1 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.

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

Provides a "mapping" of variables to responses using a simulation. Protected due to Interface letter-envelope idiom.

The function evaluator for application interfaces. Called from derived_compute_response() and derived_asynch_compute_response() in derived Model classes. If asynch_flag is not set, perform a blocking evaluation (using derived_map()). If asynch_flag is set, add the job to the 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.

10.6.2.3 const ResponseArray & synch () `[protected, virtual]`

executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPList 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_synchronize() in derived Model classes.

Reimplemented from Interface.

10.6.2.4 const IntResponseMap & synch_nowait () `[protected, virtual]`

executes a nonblocking schedule for asynchronous evaluations in the beforeSynchCorePRPList 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_synchronize_nowait() in derived Model classes.

Reimplemented from Interface.

10.6.2.5 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.

10.6.2.6 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.

10.6.2.7 void self_schedule_analyses () `[protected]`

blocking self-schedule of all analyses within a function evaluation using message passing

This code is called from derived classes to provide the master portion of a master-slave algorithm for the dynamic 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.

10.6.2.8 void serve_analyses_synch () `[protected]`

serve the master analysis scheduler and manage one synchronous analysis job at a time

This code is called from derived classes to run synchronous analyses on slave processors. The slaves receive requests (blocking receive), do local derived_map_ac's, and return codes. This is done continuously until a termination signal is received from the master. It is patterned after serve_evaluations_synch().

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

checks data_pairs and beforeSynchCorePRPList to see if the current evaluation request has already been performed or queued

Called from map() to check incoming evaluation request for duplication with content of data_pairs and beforeSynchCorePRPList. If duplication is detected, return true, else return false. Manage bookkeeping with historyDuplicateMap and beforeSynchDuplicateMap. Note that the list searches can get very expensive if a long list is searched on every new function evaluation (either from a large number of previous jobs, a large number of pending jobs, or both). For this reason, a user request for deactivation of the evaluation cache results in a complete bypass of duplication_detect(), even though a beforeSynchCorePRPList search would still be meaningful. Since the intent of this request is to streamline operations, both list searches are bypassed.

10.6.2.10 void self_schedule_evaluations () `[private]`

blocking self-schedule of all evaluations in beforeSynchCorePRPList 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- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

10.6.2.11 void static_schedule_evaluations () `[private]`

blocking static schedule of all evaluations in beforeSynchCorePRPList 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.

10.6.2.12 void asynchronous_local_evaluations (PRPList & *prp_list*) `[private]`

perform all jobs in prp_list using asynchronous approaches on the local processor

This function provides blocking synchronization for the local asynch case (background system call, nonblocking fork, or threads). It can be called from synch() for a complete local scheduling of all asynchronous jobs or from static_schedule_evaluations() to perform a local portion of the total job set. It uses the derived_map_asynch() to initiate asynchronous evaluations and derived_synch() to capture completed jobs, and mirrors the self_schedule_evaluations() message passing scheduler as much as possible (derived_synch() is modeled after MPI_Waitsome()).

10.6.2.13 void synchronous_local_evaluations (PRPList & *prp_list*) `[private]`

perform all jobs in prp_list using synchronous approaches on the local processor

This function provides blocking synchronization for the local synchronous case (foreground system call, blocking fork, or procedure call from derived_map()). It is called from static_schedule_evaluations() to perform a local portion of the total job set.

10.6.2.14 void asynchronous_local_evaluations_nowait (PRPList & *prp_list*) `[private]`

launch new jobs in prp_list asynchronously (if capacity is available), perform nonblocking query of all running jobs, and 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 synch_nowait() and passed the complete set of all asynchronous jobs (beforeSynchCorePRPList). It uses derived_map_asynch() to initiate asynchronous evaluations and derived_synch_nowait() to capture completed jobs in nonblocking mode. It mirrors a nonblocking message passing scheduler as much as possible (derived_synch_nowait() modeled after MPI_Testsome()). The result of this function is rawResponseMap, which uses fn_eval_id as a key. It is assumed that the incoming prp_list contains only active and new jobs - i.e., all completed jobs are cleared by synch_nowait().

10.6.2.15 void serve_evaluations_synch () `[private]`

serve the evaluation message passing schedulers and perform one synchronous evaluation at a time

This code is invoked by serve_evaluations() to perform one synchronous job at a time on each slave/peer server. The servers receive requests (blocking receive), do local synchronous maps, and return results. This is done continuously until a termination signal is received from the master (sent via stop_evaluation_servers()).

10.6.2.16 void serve_evaluations_asynch () `[private]`

serve the evaluation message passing schedulers and manage multiple asynchronous evaluations

This code is invoked by serve_evaluations() to perform multiple asynchronous jobs on each slave/peer server. The servers test for any incoming jobs, launch any new jobs, process any completed jobs, and return any results. Each of these components is nonblocking, although the server loop continues until a termination signal is received from the master (sent via stop_evaluation_servers()). In the master-slave case, the master maintains the correct number of jobs on each slave. In the static scheduling case, each server is responsible for limiting concurrency (since the entire static schedule is sent to the peers at start up).

10.6.2.17 void serve_evaluations_peer () `[private]`

serve the evaluation message passing schedulers and perform one synchronous evaluation at a time as part of the 1st peer

This code is invoked by serve_evaluations() to perform a synchronous evaluation in coordination with the iteratorCommRank 0 processor (the iterator) for static schedules. The bcast() matches either the bcast() in synchronous_local_evaluations(), which is invoked by static_schedule_evaluations()), or the bcast() in map().

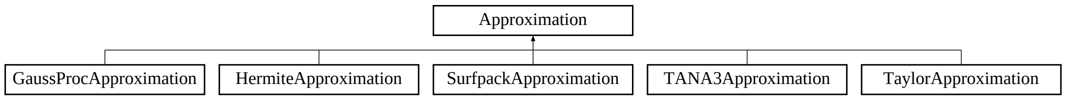
The documentation for this class was generated from the following files:

- ApplicationInterface.H
- ApplicationInterface.C

10.7 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_acv)

  standard constructor for envelope

- Approximation (const String &approx_type, const size_t &num_acv)

  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 RealVector & approximation_coefficients ()

  return the coefficient array computed by find_coefficients()

- virtual int num_coefficients () const

  return the minimum number of samples required to build the derived class approximation type in numVars dimensions

- virtual int num_constraints () const

  return the number of constraints to be enforced via anchorPoint

- virtual void clear_current ()

  clear current build data in preparation for next build

- virtual void second_order_flag (bool flag)

  set the Approximation's secondOrderFlag, if present

- int required_samples (bool constraint_flag) const

  return the minimum number of samples required to build the approximation type in numVars dimensions. Uses num_coefficients() and num_constraints().

- int num_variables () const

  return the number of variables used in the approximation

- void update (const RealVectorArray &c_vars_samples, const ResponseArray &resp_samples, const int &fn_index)

  populates currentPoints

- void update (const RealVector &c_vars, const Response &response, const int &fn_index)

  populates anchorPoint

- void update (const RealVector &c_vars, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)

  populates anchorPoint

- void append (const RealVector &c_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 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 ()

  render the approximate surface using the 3D graphics (2 variable problems only).

Protected Member Functions

- Approximation (BaseConstructor, ProblemDescDB &problem_db, const size_t &num_acv)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- virtual void find_coefficients ()

  calculate the data fit coefficients using currentPoints and anchorPoint

Protected Attributes

- bool useGradsFlag

  flag signaling the use of gradient data in global approximation builds as indicated by the user's `use_gradients` *specification. This setting cannot be inferred from the responses spec., since we may need gradient support for evaluating gradients at a single point (e.g., the center of a trust region), but not require gradient evaluations at every point.*

- bool verboseFlag

  flag for verbose approximation output

- int numVars

  number of variables in the approximation

- String approxType

  approximation type identifier

- 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().

- List< SurrogateDataPoint > currentPoints *list of samples used to build the approximation. These sample points may be fit approximately (based on a least squares regression).*

- SurrogateDataPoint anchorPoint

  a special sample (often at the center of the approximation region) for which exact matching is enforced (e.g., using equality-constrained least squares)

Private Member Functions

- Approximation ∗ get_approx (ProblemDescDB &problem_db, const size_t &num_acv)

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

- Approximation ∗ get_approx (const String &approx_type, const size_t &num_acv)

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

- void add (const RealVector &c_vars, const Response &response, const int &fn_index, bool anchor_flag)

  add a new data point by either appending to currentPoints or assigning to anchorPoint, as dictated by anchor_flag. Uses add_point() and add_anchor().

- 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

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

10.7.1 Detailed Description

Base class for the approximation class hierarchy.

The Approximation class is the base class for the data fit surrogate 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.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 Approximation ()

default constructor

The default constructor is used in List<Approximation> instantiations and by the alternate envelope constructor. approxRep is NULL in this case (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.

10.7.2.2 Approximation (ProblemDescDB & *problem_db*, const size_t & *num_acv*)

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.

10.7.2.3 Approximation (const String & *approx_type*, const size_t & *num_acv*)

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.

10.7.2.4 Approximation (const Approximation & *approx*)

copy constructor

Copy constructor manages sharing of approxRep and incrementing of referenceCount.

10.7.2.5 ~Approximation () `[virtual]`

destructor

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

10.7.2.6 Approximation (BaseConstructor, ProblemDescDB & *problem_db*, const size_t & *num_acv*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_approx() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_approx() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ∼Approximation).

10.7.3 Member Function Documentation

10.7.3.1 Approximation operator= (const Approximation & *approx*)

assignment operator

Assignment operator decrements referenceCount for old approxRep, assigns new approxRep, and increments referenceCount for new approxRep.

10.7.3.2 void clear_current () `[virtual]`

clear current build data in preparation for next build

Redefined by TANA3Approximation to clear current data but preserve history.

Reimplemented in TANA3Approximation.

10.7.3.3 void second_order_flag (bool *flag*) `[virtual]`

set the Approximation's secondOrderFlag, if present

Redefined by TaylorApproximation to set secondOrderFlag.

Reimplemented in TaylorApproximation.

10.7.3.4 void clear_all ()

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).

10.7.3.5 Approximation ∗ get_approx (ProblemDescDB & *problem_db*, const size_t & *num_acv*) `[private]`

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

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

10.7.3.6 Approximation ∗ get_approx (const String & *approx_type*, const size_t & *num_acv*) `[private]`

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

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

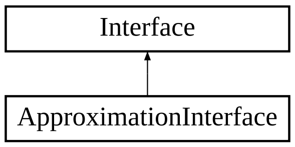
The documentation for this class was generated from the following files:

- DakotaApproximation.H
- DakotaApproximation.C

10.8 ApproximationInterface Class Reference

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

Inheritance diagram for ApproximationInterface::



Public Member Functions

- ApproximationInterface (ProblemDescDB &problem_db, const size_t &num_acv, const size_t &num_fns)

  constructor
- ∼ApproximationInterface ()

  destructor

Protected Member Functions

- void map (const Variables &vars, const ActiveSet &set, Response &response, const bool asynch_-flag=false)

  the function evaluator: provides an approximate "mapping" from the variables to the responses using functionSurfaces
- int minimum_samples (bool constraint_flag) const

  returns the minimum number of samples required to build the functionSurfaces
- void update_approximation (const RealVectorArray &all_variables, const ResponseArray &all_responses)

  passes multiple points to an approximation for building a surrogate
- void update_approximation (const RealVector &c_variables, const Response &response)
- void build_approximation (const RealVector &lower_bnds, const RealVector &upper_bnds)
- void append_approximation (const RealVector &c_variables, const Response &response)

  updates an existing global approximation with new data
- void clear ()

  clears all data from an approximation interface

- bool anchor () const

  queries the presence of an anchorPoint within an approximation interface

- const RealVectorArray & approximation_coefficients ()

  retrieve the approximation coefficients from each Approximation 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

- bool completeApproxSet

  flag for complete approximation set (no mixture of truth/approx responses at Model level)

- IntArray approxFnIds

  for incomplete approximation sets, this array specifies the response function subset that is approximated

- Array< Approximation > functionSurfaces

  list of approximations, one per response function

- RealVectorArray functionSurfaceCoeffs

  array of approximation coefficient vectors, one vector per response function

- bool graphicsFlag

  controls 3D graphics of approximation surfaces

- IntResponseMap beforeSynchResponseMap

  bookkeeping map to catalogue responses generated in map() for use in synch() and synch_nowait(). This supports pseudo-asynchronous operations (approximate responses are always computed synchronously, but asynchronous virtual functions are supported through bookkeeping).

10.8.1 Detailed Description

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

ApproximationInterface provides an interface class for building a set of global/local/multipoint approximations and performing approximate function evaluations using them. It contains a list of Approximation objects, one for each response function.

10.8.2 Member Function Documentation

10.8.2.1 void update_approximation (const RealVector & *c_variables*, const Response & *response*) `[protected, virtual]`

Evaluate values, gradients, and possibly Hessians at the current point for building a local approximation.

Reimplemented from Interface.

10.8.2.2 void build_approximation (const RealVector & *lower_bnds*, const RealVector & *upper_bnds*) `[protected, virtual]`

Evaluate values, gradients, and possibly Hessians at the current point for building a local approximation.

Reimplemented from Interface.

10.8.3 Member Data Documentation

10.8.3.1 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

10.9 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 & operator[ ] (size_t i)

  Index operator, returns the ith value of the array.

- const T & operator[ ] (size_t i) const

  Index operator const, returns the ith value of the array.

- T & operator() (size_t i)

  Index operator, not bounds checked.

- const T & operator() (size_t i) const

  Index operator const, not bounds checked.

- void read (istream &s)

  Reads an Array from an istream.

- void write (ostream &s) const

  Writes an Array to an output stream.

- void write (ostream &s, const Array< String > &label_array) const

  Writes an Array and associated label array to an output stream.

- void write_aprepro (ostream &s, const Array< String > &label_array) const

  Writes an Array and associated label array to an output stream in aprepro format.

- void write_annotated (ostream &s, bool write_len) const

  Writes an Array to an output stream in annotated format.

- void read (BiStream &s)

  Reads an Array from a binary input stream.

- void write (BoStream &s) const

  Writes an Array to a binary output stream.

- void read (MPIUnpackBuffer &s)

  Reads an Array from a buffer after an MPI receive.

- void write (MPIPackBuffer &s) const

  Writes an Array to a buffer prior to an MPI send.

- size_t length () const

  Returns size of array.

- void reshape (size_t sz)

  Resizes array to size sz.

- 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.

10.9.1 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 inherite from either the STL or RW vector classes.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 Array (const T ∗ *p*, size_t *size*) `[inline]`

Constructor which copies size entries from T∗.

Assigns size values from p into array.

10.9.3 Member Function Documentation

10.9.3.1 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.

10.9.3.2 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.

10.9.3.3]

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).

10.9.3.4]

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).

10.9.3.5 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

10.9.3.6 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

10.9.3.7 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

10.10 BaseConstructor Struct Reference

Dummy struct for overloading letter-envelope constructors.

Public Member Functions

- BaseConstructor (int=0)

  C++ structs can have constructors.

10.10.1 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.

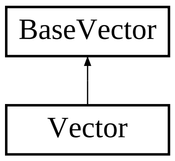
The documentation for this struct was generated from the following file:

- global_defs.h

10.11 BaseVector Class Template Reference

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

Inheritance diagram for BaseVector::



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.

10.11.1 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.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 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

10.11.3 Member Function Documentation

10.11.3.1]

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).

10.11.3.2]

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).

10.11.3.3 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

10.11.3.4 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

10.11.3.5 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

10.11.3.6 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

10.11.3.7 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.

10.11.3.8 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

10.12 BiStream Class Reference

The binary input stream class. Overloads the >> operator for all data types.

Public Member Functions

- BiStream ()

  Default constructor, need to open.

- BiStream (const char ∗s)

  Constructor takes name of input file.

- BiStream (const char ∗s, std::ios_base::openmode mode)

  Constructor takes name of input file, mode.

- BiStream (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.

10.12.1 Detailed Description

The binary input stream class. Overloads the >> operator for all data types.

The Dakota::BiStream class is a binary input class which overloads the >> operator for all standard data types (int, char, float, etc). The class relies on the methods within the ifstream base class. The Dakota::BiStream class inherits from the ifstream class. If available, the class utilize rpc/xdr to construct machine independent binary files. These Dakota restart files can be moved from host to host. The motivation to develop these classes was to replace the Rogue wave classes which Dakota historically used for binary I/O.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 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.

10.12.2.2 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.

10.12.2.3 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.

10.12.2.4 ∼BiStream ()

Destructor, calls xdr_destroy to delete xdr stream.

Destructor, destroys the xdr stream allocated in constructor

10.12.3 Member Function Documentation

10.12.3.1 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.

10.12.3.2 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

10.13 BoStream Class Reference

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

Public Member Functions

- BoStream ()

  Default constructor, need to open.

- BoStream (const char ∗s)

  Constructor takes name of input file.

- BoStream (const char ∗s, std::ios_base::openmode mode)

  Constructor takes name of input file, mode.

- 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.

10.13.1 Detailed Description

The binary output stream class. Overloads the << operator for all 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.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 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.

10.13.2.2 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

10.13.2.3 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.

10.13.3 Member Function Documentation

10.13.3.1 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.

10.13.3.2 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

10.14 COLINApplication Class Reference

Public Member Functions

- COLINApplication (Model &model, Optimizer ∗opt_)

  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 number of 'slave' processors that can perform evaluations. 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 dakota_asynch_flag (const bool &asynch_flag)

  This function publishes the iterator's asynchFlag at run time (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 & userDefinedModel

  reference to the COLINOptimizer's model passed in the constructor
- ActiveSet activeSet

  copy/conversion of the COLIN request vector
- bool dakotaModelAsynchFlag

a flag for asynchronous DAKOTA evaluations

- bool blockingSynch

  flag for user specification of "synchronization blocking"

- IntResponseMap dakotaResponseMap

  map of DAKOTA responses returned by synchronize_nowait()

- size_t numObjFns

  number of objective functions

- size_t numNonlinCons

  number of nonlinear constraints

- Optimizer ∗ opt

  pointer to the DAKOTA Optimizer hierarchy passed through the COLINApplication constructor. This is needed for accessing Optimizer functions (e.g., multi_objective_modify()) needed by COLINApplication.

- 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

- std::list< int > requested_async_id

  tracks COLIN response ids from DoEval() to next_eval()

10.14.1 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.

10.14.2 Member Function Documentation

10.14.2.1 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).

10.14.2.2 void synchronize ()

blocking retrieval of all pending jobs

Blocking synchronize of asynchronous DAKOTA jobs followed by conversion of the Response objects to ColinResponse response objects.

10.14.2.3 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.

10.14.2.4 void map_response (ColinResponse & *colin_response*, const Response & *dakota_response*) `[private]`

utility function for mapping a DAKOTA response to a COLIN response

map_response Maps a Response object into a ColinResponse class that is compatable with COLIN.

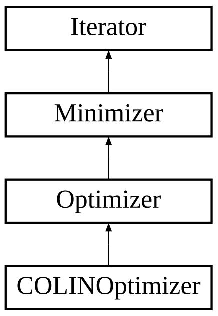
The documentation for this class was generated from the following files:

- COLINApplication.H
- COLINApplication.C

10.15 COLINOptimizer Class Template Reference

Wrapper class for optimizers defined using COLIN.

Inheritance diagram for COLINOptimizer::



Public Member Functions

- COLINOptimizer (Model &model)

  Section 2

- ~COLINOptimizer ()

  destructor

- void find_optimum (void)

  Performs the iterations to determine the optimal solution.

Protected Member Functions

- virtual void set_rng (void)

  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 (void)

  sets options for specific methods based on user specifications (called at construction time)

- void set_standard_method_parameters (void)

  sets the standard method parameters shared by all methods

- virtual void set_runtime_parameters (void)

  sets method parameters for specific methods using data that is not available until run time

Protected Attributes

- OptimizerT ∗ optimizer

  Pointer to COLIN base optimizer object.

- COLINApplication ∗ application

  Pointer to the COLINApplication object.

- OptProblem< ColinPoint > problem

  the COLIN problem object

- utilib::RNG ∗ rng

  RNG ptr.

- String evalSynch

  the `synchronization` *setting (*`blocking` *or* `nonblocking`*)*

10.15.1 Detailed Description

template<class OptimizerT> class Dakota::COLINOptimizer< OptimizerT >

Wrapper class for optimizers defined using COLIN.

The COLINOptimizer 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. COLINOptimizer 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. COLIN methods assume asynchronous operations whenever the algorithm has independent evaluations which can be performed simultaneously (implicit parallelism). Therefore, parallel configuration is not mapped into the method, rather it is used in COLINApplication to control whether or not an asynchronous evaluation request from the method is honored by the model (exception: pattern search exploratory moves is set to `best_all` for parallel function evaluations). Refer to [Hart, W.E., 1997] for additional information on COLIN objects and controls.

10.15.2 Member Function Documentation

10.15.2.1 void find_optimum (void) `[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.

10.15.2.2 void set_standard_method_parameters (void) `[protected]`

sets the standard method parameters shared by all methods

set_standard_method_parameters propagates standard DAKOTA user input to the optimizer.

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

- COLINOptimizer.H

10.16 ColinPoint Class Reference

Public Attributes

- vector< double > rvec

    continuous parameter values

- vector< int > ivec

    discrete parameter values

10.16.1 Detailed Description

A class containing a vector of doubles and integers.

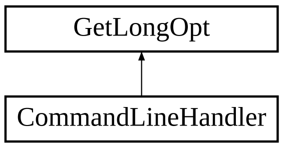
The documentation for this class was generated from the following file:

- COLINApplication.H

10.17 CommandLineHandler Class Reference

Utility class for managing command line inputs to DAKOTA.

Inheritance diagram for CommandLineHandler::



Public Member Functions

- CommandLineHandler ()

  default constructor, requires check_usage() call for parsing

- CommandLineHandler (int argc, char ∗∗argv)

  constructor with parsing

- ∼CommandLineHandler ()

  destructor

- void check_usage (int argc, char ∗∗argv)

  Verifies that DAKOTA is called with the correct command usage. Prints a descriptive message and exits the program if incorrect.

- int read_restart_evals () const

  Returns the number of evaluations to be read from the restart file (as specified on the DAKOTA command line) as an integer instead of a const char∗.

Private Member Functions

- void initialize_options ()

  enrolls the supported command line inputs.

- void output_version (ostream &s) const

  outputs the DAKOTA version

10.17.1 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

10.18 CommandShell Class Reference

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

Public Member Functions

- CommandShell ()
  constructor
- ~CommandShell ()
  destructor
- CommandShell & operator<< (const char ∗string)
  adds string to unixCommand
- CommandShell & operator<< (CommandShell &(∗f)(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
  the command string that is constructed through one or more << insertions and then executed by flush
- bool asynchFlag
  flags nonblocking operation (background system calls)

- bool suppressOutputFlag

  flags suppression of shell output (no command echo)

10.18.1 Detailed Description

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

The CommandShell class wraps the C system() utility and defines convenience operators for building a command string and then passing it to the shell.

10.18.2 Member Function Documentation

10.18.2.1 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.

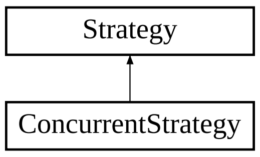
The documentation for this class was generated from the following files:

- CommandShell.H
- CommandShell.C

10.19 ConcurrentStrategy Class Reference

Strategy for multi-start iteration or pareto set optimization.

Inheritance diagram for ConcurrentStrategy::



Public Member Functions

- ConcurrentStrategy (ProblemDescDB &problem_db)

  constructor

- ~ConcurrentStrategy ()

  destructor

- void run_strategy ()

  Performs the concurrent strategy by executing selectedIterator on userDefinedModel multiple times in parallel for different settings within the iterator or model.

Private Member Functions

- void self_schedule_iterators ()

  executed by the strategy master to self-schedule iterator jobs among slave iterator servers (called by run_strategy())

- void serve_iterators ()

  executed on the slave iterator servers to perform iterator jobs assigned by the strategy master (called by run_strategy())

- void static_schedule_iterators ()

  executed on iterator peers to statically schedule iterator jobs (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

  an array of parameter set vectors (either multistart variable 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

10.19.1 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 mutiobjective formulations) is provided. This pareto set is mapped through running an optimizer multiple times for different sets of multiobjective weightings.

10.19.2 Member Function Documentation

10.19.2.1 void self_schedule_iterators () `[private]`

executed by the strategy master to self-schedule iterator jobs among slave iterator servers (called by run_strategy())

This function is adapted from ApplicationInterface::self_schedule_evaluations().

10.19.2.2 void serve_iterators () `[private]`

executed on the slave iterator servers to perform iterator jobs assigned by the strategy master (called by run_strategy())

This function is similar in structure to ApplicationInterface::serve_evaluations_synch().

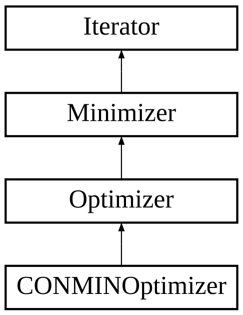
The documentation for this class was generated from the following files:

- ConcurrentStrategy.H
- ConcurrentStrategy.C

10.20 CONMINOptimizer Class Reference

Wrapper class for the CONMIN optimization library.

Inheritance diagram for CONMINOptimizer::



Public Member Functions

- CONMINOptimizer (Model &model)

  constructor

- ~CONMINOptimizer ()

  destructor

- void find_optimum ()

  Used within the optimizer branch for computing the optimal solution. Redefines the run_iterator virtual function for the optimizer branch.

Protected Member Functions

- virtual void derived_pre_run ()

  performs run-time set up

- virtual void derived_post_run ()

  performs final solution processing

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

- SizetList constraintMappingIndices

  a list of indices for referencing the corresponding Response constraints used in computing the CONMIN constraints.

- RealList constraintMappingMultipliers

  a list of multipliers for mapping the Response constraints to the CONMIN constraints.

- RealList constraintMappingOffsets

  a list of offsets for mapping the Response constraints to the CONMIN constraints.

- int N1

  Size variable for CONMIN arrays. See CONMIN manual.

- int N2

  Size variable for CONMIN arrays. See CONMIN manual.

- int N3

  Size variable for CONMIN arrays. See CONMIN manual.

- int N4

  Size variable for CONMIN arrays. See CONMIN manual.

- int N5

  Size variable for CONMIN arrays. See CONMIN manual.

- int NFDG

  Finite difference flag.

- int IPRINT

  Flag to control amount of output data.

- int ITMAX

  Flag to specify the maximum number of iterations.

- double FDCH

  Relative finite difference step size.

- double FDCHM

  Absolute finite difference step size.

- double CT

  Constraint thickness parameter.

- double CTMIN

  Minimum absolute value of CT used during optimization.

- double CTL

  Constraint thickness parameter for linear and side constraints.

- double CTLMIN

  Minimum value of CTL used during optimization.

- double DELFUN

  Relative convergence criterion threshold.

- double DABFUN

  Absolute convergence criterion threshold.

- double ∗ conminDesVars

  Array of design variables used by CONMIN (length 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.

10.20.1 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.

10.20.2 Member Data Documentation

10.20.2.1 int conminInfo `[private]`

INFO from CONMIN manual.

Information requested by CONMIN: 1 = evaluate objective and constraints, 2 = evaluate gradients of objective and constraints.

10.20.2.2 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

10.20.2.3 int optimizationType `[private]`

MINMAX from DOT manual (minimize or maximize).

Values of 0 or -1 (minimize) or 1 (maximize).

10.20.2.4 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 <= 0 (which requires a transformation from 2-sided inequalities and equalities).

10.20.2.5 SizetList constraintMappingIndices `[private]`

a list of indices for referencing the corresponding 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.

10.20.2.6 RealList constraintMappingMultipliers `[private]`

a list of multipliers for mapping the Response constraints to the CONMIN constraints.

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.

10.20.2.7 RealList constraintMappingOffsets `[private]`

a list of offsets for mapping the Response constraints to the 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.

10.20.2.8 int N1 `[private]`

Size variable for CONMIN arrays. See CONMIN manual.

N1 = number of variables + 2

10.20.2.9 int N2 `[private]`

Size variable for CONMIN arrays. See CONMIN manual.

N2 = number of constraints + 2∗(number of variables)

10.20.2.10 int N3 `[private]`

Size variable for CONMIN arrays. See CONMIN manual.

N3 = Maximum possible number of active constraints.

10.20.2.11 int N4 `[private]`

Size variable for CONMIN arrays. See CONMIN manual.

N4 = Maximum(N3,number of variables)

10.20.2.12 int N5 `[private]`

Size variable for CONMIN arrays. See CONMIN manual.

N5 = 2∗(N4)

10.20.2.13 double CT `[private]`

Constraint thickness parameter.

The value of CT decreases in magnitude during optimization.

10.20.2.14 double* S `[private]`

Internal CONMIN array.

Move direction in N-dimensional space.

10.20.2.15 double* G1 `[private]`

Internal CONMIN array.

Temporary storage of constraint values.

10.20.2.16 double* G2 `[private]`

Internal CONMIN array.

Temporary storage of constraint values.

10.20.2.17 double* B `[private]`

Internal CONMIN array.

Temporary storage for computations involving array S.

10.20.2.18 double* C `[private]`

Internal CONMIN array.

Temporary storage for use with arrays B and S.

10.20.2.19 int* MS1 `[private]`

Internal CONMIN array.

Temporary storage for use with arrays B and S.

10.20.2.20 double* SCAL `[private]`

Internal CONMIN array.

Vector of scaling parameters for design parameter values.

10.20.2.21 double* DF [private]

Internal CONMIN array.

Temporary storage for analytic gradient data.

10.20.2.22 double* A [private]

Internal CONMIN array.

Temporary 2-D array for storage of constraint gradients.

10.20.2.23 int* ISC [private]

Internal CONMIN array.

Array of flags to identify linear constraints. (not used in this implementation of CONMIN)

10.20.2.24 int* IC [private]

Internal CONMIN array.

Array of flags to identify active and violated constraints

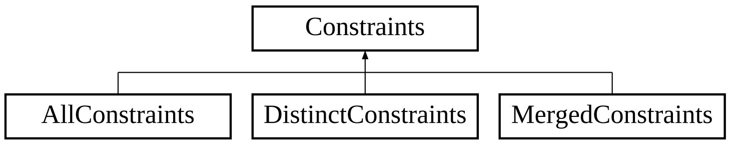
The documentation for this class was generated from the following files:

- CONMINOptimizer.H
- CONMINOptimizer.C

10.21 Constraints Class Reference

Base class for the variable constraints class hierarchy.

Inheritance diagram for Constraints::



Public Member Functions

- Constraints ()

  default constructor

- Constraints (const ProblemDescDB &problem_db, const pair< short, short > &view)

  standard constructor

- 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 RealVector all_continuous_upper_bounds () const

    returns a single array with all continuous upper bounds

- virtual IntVector all_discrete_lower_bounds () const

    returns a single array with all discrete lower bounds

- virtual IntVector all_discrete_upper_bounds () const

    returns a single array with all discrete upper bounds

- 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

  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

Protected Member Functions

- Constraints (BaseConstructor, const ProblemDescDB &problem_db, const pair< short, short > &view)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- void manage_linear_constraints (const ProblemDescDB &problem_db)

  perform checks on user input, convert linear constraint coefficient input to matrices, and assign defaults

Protected Attributes

- pair< short, short > variablesView

  the variables view pair containing active (first) and inactive (second) 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

  an empty real vector returned in get functions when there are no variable constraints corresponding to the request

- IntVector emptyIntVector

  an empty int vector returned in get functions when there are no variable constraints corresponding to the request

Private Member Functions

- Constraints ∗ get_constraints (const ProblemDescDB &problem_db, const pair< short, short > &view)

  Used only by the constructor to initialize constraintsRep to the appropriate derived type.

Private Attributes

- Constraints ∗ constraintsRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing constraintsRep

10.21.1 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.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 Constraints ()

default constructor

The default constructor: constraintsRep is NULL in this case (a populated problem_db is needed to build a meaningful Constraints object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

10.21.2.2 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.

10.21.2.3 Constraints (const Constraints & *con*)

copy constructor

Copy constructor manages sharing of constraintsRep and incrementing of referenceCount.

10.21.2.4 ~Constraints () `[virtual]`

destructor

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

10.21.2.5 Constraints (BaseConstructor, const ProblemDescDB & *problem_db*, const pair< short, short > & *view*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_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).

10.21.3 Member Function Documentation

10.21.3.1 Constraints operator= (const Constraints & *con*)

assignment operator

Assignment operator decrements referenceCount for old constraintsRep, assigns new constraintsRep, and increments referenceCount for new constraintsRep.

10.21.3.2 void manage_linear_constraints (const ProblemDescDB & *problem_db*) `[protected]`

perform checks on user input, convert linear constraint coefficient input to matrices, and assign defaults

Convenience function called from derived class constructors. The number of variables active for applying linear constraints is currently defined to be the number of active continuous variables plus the number of active discrete variables (the most general case), even though very few optimizers can currently support mixed variable linear constraints.

10.21.3.3 Constraints ∗ get_constraints (const ProblemDescDB & *problem_db*, const pair< short, short > & *view*) `[private]`

Used only by the constructor to initialize constraintsRep to the appropriate derived type.

Initializes constraintsRep to the appropriate derived type, as given by the variables view.

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

- DakotaConstraints.H
- DakotaConstraints.C

10.22 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** }

  Error codes reported by the engine - Most of these codes never really 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.

- std::string match (const std::string &str)

  matches a particular string; this method returns a string that is a sub-string matching with the regular expression

- bool match (const std::string &str, size_t ∗start, size_t ∗size)

  another form of matching; returns the indexes of the maching

- 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)

Split.

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.

10.22.1 Detailed Description

DESCRIPTION: Wrapper for the Regular Expression engine(regexp) released by Henry Spencer of the University of Toronto.

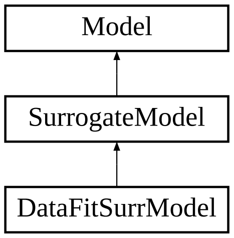
The documentation for this class was generated from the following files:

- CtelRegExp.H
- CtelRegExp.C

10.23 DataFitSurrModel Class Reference

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

Inheritance diagram for DataFitSurrModel::



Public Member Functions

- DataFitSurrModel (ProblemDescDB &problem_db)

  constructor
- ∼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)

- Interface & interface ()

  return approxInterface

- void surrogate_bypass (bool bypass_flag)

  set surrogateBypass flag and pass request on to actualModel for any lower-level surrogates.

- void build_approximation ()

  Builds the local/multipoint/global approximation using daceIterator/actualModel.

- bool build_approximation (const RealVector &c_vars, const Response &response)

  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data that augments the c_vars/response anchor point.

- void update_approximation (const RealVector &c_vars, const Response &response)

  Adds a point to a global approximation and rebuilds it (requests forwarded to approxInterface).

- const RealVectorArray & approximation_coefficients ()

  return the approximation coefficients from each Approximation (request forwarded to approxInterface)

- void component_parallel_mode (int mode)

  update component parallel mode for supporting parallelism in actualModel

- void derived_init_communicators (const int &max_iterator_concurrency)

  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)

  set active parallel configuration within actualModel

- void reset_communicators ()

  reset communicator partitions for the DataFitSurrModel (request forwarded to actualModel)

- void derived_free_communicators (const int &max_iterator_concurrency)

  deallocate communicator partitions for the DataFitSurrModel (request forwarded to actualModel)

- void serve ()

Service actualModel job requests received from the master. Completes when a termination message is received from stop_servers().

- void stop_servers ()

  Executed by the master to terminate actualModel server operations when DataFitSurrModel iteration is complete.

- int evaluation_id () const

  return the current evaluation id for the DataFitSurrModel

- void set_evaluation_reference ()

  set the evaluation counter reference points for the DataFitSurrModel (request forwarded to approxInterface and actualModel)

- void print_evaluation_summary (ostream &s, bool minimal_header=false, bool relative_count=true) const

  print the evaluation summary for the DataFitSurrModel (request forwarded to approxInterface and actualModel)

Private Member Functions

- void update_actual_model ()

  update actualModel with current variable values/bounds/labels

- 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 asv_mapping (const IntArray &orig_asv, IntArray &actual_asv, IntArray &approx_asv, bool build_flag)

  distributes the incoming orig_asv among actual_asv and approx_asv

Private Attributes

- int surrModelEvals

  number of calls to derived_compute_response()/ derived_asynch_compute_response()

- String sampleReuse

user selection of type of sample reuse for approximation builds: all, region, file, or none (default)

- String sampleReuseFile

  file name for sampleReuse == "file"

- Interface approxInterface

  manages the building and subsequent evaluation of the approximations (required for both global and local)

- String actualModelPointer

  string identifier for the actual model from the local/multipoint approximation specification; used to build actual-Model.

- Model actualModel

  the truth model which provides evaluations for building the surrogate (optional for global, required for local)

- String daceMethodPointer

  string identifier for the dace method from the global approximation specification; used in building daceIterator and actualModel for global approximations (optional for global since restart data may also be used)

- Iterator daceIterator

  selects parameter sets on which to evaluate actualModel in order to generate the necessary data for building global approximations (optional for global since restart data may also be used)

10.23.1 Detailed Description

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

The DataFitSurrModel class manages global or local approximations (surrogates that involve data fits) that are used in place of an expensive model. The class contains an approxInterface (required for both global and local) which manages the approximate function evaluations, an actualModel (optional for global, required for local) which provides truth evaluations for building the surrogate, and a daceIterator (optional for global, not used for local) which selects parameter sets on which to evaluate actualModel in order to generate the necessary data for building global approximations.

10.23.2 Member Function Documentation

10.23.2.1 void derived_compute_response (const ActiveSet & *set*) `[protected, virtual]`

portion of compute_response() specific to DataFitSurrModel

Build the approximation (if needed), evaluate the approximate response using approxInterface, and, if correction is active, correct the results.

Reimplemented from Model.

10.23.2.2 void derived_asynch_compute_response (const ActiveSet & *set*) `[protected, virtual]`

portion of asynch_compute_response() specific to DataFitSurrModel

Build the approximation (if needed) and evaluate the approximate response using approxInterface in a quasi-asynchronous approach (ApproximationInterface::map() performs the map synchronously and bookkeeps the results for return in derived_synchronize() below).

Reimplemented from Model.

10.23.2.3 const ResponseArray & derived_synchronize () `[protected, virtual]`

portion of synchronize() specific to DataFitSurrModel

Retrieve quasi-asynchronous evaluations from approxInterface and, if correction is active, apply correction to each response in the array.

Reimplemented from Model.

10.23.2.4 const IntResponseMap & derived_synchronize_nowait () `[protected, virtual]`

portion of synchronize_nowait() specific to DataFitSurrModel

Retrieve quasi-asynchronous evaluations from approxInterface and, if correction is active, apply correction to each response in the map.

Reimplemented from Model.

10.23.2.5 void derived_init_communicators (const int & *max_iterator_concurrency*) `[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.

10.23.2.6 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.

10.23.2.7 void update_actual_model () `[private]`

update actualModel with current variable values/bounds/labels

Update variables and constraints data within actualModel using values and labels from currentVariables and bound/linear/nonlinear constraints from userDefinedConstraints.

10.23.2.8 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).

10.23.2.9 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.

10.23.3 Member Data Documentation

10.23.3.1 String actualModelPointer `[private]`

string identifier for the actual model from the local/multipoint approximation specification; used to build actualModel.

Specification is used only for local/multipoint approximations, since the dace_method_pointer in the global approximation specification is responsible for identifying all actualModel components.

10.23.3.2 Model actualModel `[private]`

the truth model which provides evaluations for building the surrogate (optional for global, required for local)

actualModel is unrestricted in type; arbitrary nestings are possible.

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

- DataFitSurrModel.H
- DataFitSurrModel.C

10.24 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

    string identifier for an interface specification data set (from the id_interface specification in **InterfIndControl***)*

- String interfaceType

    the interface selection: system, fork, direct, or grid

- String algebraicMappings

    defines the subset of the parameter to response mapping that is explicit and algebraic. This is typically a stub.nl filename (AMPL format) from JAGUAR.

- StringArray analysisDrivers

  the set of analysis drivers for a simulation-based interface (from the `analysis_drivers` *specification in* **InterfIndControl***)*

- String2DArray analysisComponents

  the set of analysis components for a simulation-based interface (from the `analysis_components` *specification in* **InterfIndControl***)*

- String inputFilter

  the input filter for a simulation-based interface (from the `input_filter` *specification in* **InterfIndControl***)*

- String outputFilter

  the output filter for a simulation-based interface (from the `output_filter` *specification in* **InterfIndControl***)*

- String parametersFile

  the parameters file for system call and fork interfaces (from the `parameters_file` *specification in* **InterfApplicSC** *and* **InterfApplicF***)*

- String resultsFile

  the results file for system call and fork interfaces (from the `results_file` *specification in* **InterfApplicSC** *and* **InterfApplicF***)*

- String analysisUsage

  the analysis command usage string for a system call interface (from the `analysis_usage` *specification in* **InterfApplicSC***)*

- bool apreproFormatFlag

  the flag for aprepro format usage in the parameters file for system call and fork interfaces (from the `aprepro` *specification in* **InterfApplicSC** *and* **InterfApplicF***)*

- bool fileTagFlag

  the flag for file tagging of parameters and results files for system call and fork interfaces (from the `file_tag` *specification in* **InterfApplicSC** *and* **InterfApplicF***)*

- bool fileSaveFlag

  the flag for saving of parameters and results files for system call and fork interfaces (from the `file_save` *specification in* **InterfApplicSC** *and* **InterfApplicF***)*

- int procsPerAnalysis

  processors per parallel analysis for a direct interface (from the `processors_per_analysis` *specification in* **InterfApplicDF***)*

- StringArray gridHostNames

  WEH - not currently used for grid computing names of host machines for a grid interface (from the `hostnames` *specification in* **InterfApplicG***).*

- IntArray gridProcsPerHost

  processors per host machine for a grid interface (from the `processors_per_host` *specification in* **InterfApplicG***)*

- String interfaceSynchronization

  parallel mode for a simulation-based interface: synchronous or asynchronous (from the `asynchronous` *specification in* **InterfIndControl***)*

- int asynchLocalEvalConcurrency

  evaluation concurrency for asynchronous simulation-based interfaces (from the `evaluation_concurrency` *specification in* **InterfIndControl***)*

- int asynchLocalAnalysisConcurrency

  analysis concurrency for asynchronous simulation-based interfaces (from the `analysis_concurrency` *specification in* **InterfIndControl***)*

- int evalServers

  number of evaluation servers to be used in the parallel configuration (from the `evaluation_servers` *specification in* **InterfIndControl***)*

- String evalScheduling

  the scheduling approach to be used for concurrent evaluations within an iterator (from the `evaluation_self_scheduling` *and evaluation_static_scheduling specifications in* **InterfIndControl***)*

- int analysisServers

  number of analysis servers to be used in the parallel configuration (from the `analysis_servers` *specification in* **InterfIndControl***)*

- String analysisScheduling

  the scheduling approach to be used for concurrent analyses within a function evaluation (from the `analysis_self_scheduling` *and* `analysis_static_scheduling` *specifications in* **InterfIndControl***)*

- String failAction

  the selected action upon capture of a simulation failure: abort, retry, recover, or continuation (from the `failure_capture` *specification in* **InterfIndControl***)*

- int retryLimit

  the limit on retries for captured simulation failures (from the `retry` *specification in* **InterfIndControl***)*

- RealVector recoveryFnVals

  the function values to be returned in a recovery operation for captured simulation failures (from the `recover` *specification in* **InterfIndControl***)*

- bool activeSetVectorFlag

  active set vector: 1=active (ASV control on), 0=inactive (ASV control off) (from the `deactivate active_set_vector` *specification in* **InterfIndControl***)*

- bool evalCacheFlag

function evaluation cache: 1=active (all new evaluations checked against existing cache and then added to cache), 0=inactive (cache neither queried nor augmented) (from the `deactivate evaluation_cache` *specification in* ***InterfIndControl****)*

- bool restartFileFlag

  function evaluation cache: 1=active (all new evaluations written to restart), 0=inactive (no records written to restart) (from the `deactivate restart_file` *specification in* ***InterfIndControl****)*

Private Member Functions

- void assign (const DataInterface &data_interface)

  convenience function for setting this objects attributes equal to the attributes of the incoming data_interface object (used by copy constructor and assignment operator)

10.24.1 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

10.25 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

  string identifier for the method specification data set (from the `id_method` *specification in* **MethodIndControl***)*
- String modelPointer

  string pointer to the model specification to be used by this method (from the `model_pointer` *specification in* **MethodIndControl***)*
- String methodOutput

method verbosity control: quiet, verbose, debug, or normal (default) (from the `output` *specification in* **MethodIndControl***)*

- int maxIterations

  maximum number of iterations allowed for the method (from the `max_iterations` *specification in* **MethodIndControl***)*

- int maxFunctionEvaluations

  maximum number of function evaluations allowed for the method (from the `max_function_evaluations` *specification in* **MethodIndControl***)*

- bool speculativeFlag

  flag for use of speculative gradient approaches for maintaining parallel load balance during the line search portion of optimization algorithms (from the `speculative` *specification in* **MethodIndControl***)*

- Real convergenceTolerance

  iteration convergence tolerance for the method (from the `convergence_tolerance` *specification in* **MethodIndControl***)*

- Real constraintTolerance

  tolerance for controlling the amount of infeasibility that is allowed before an active constraint is considered to be violated (from the `constraint_tolerance` *specification in* **MethodIndControl***)*

- bool methodScaling

  flag indicating scaling status (from the `scaling` *specification in* **MethodIndControl***)*

- RealVector linearIneqConstraintCoeffs

  coefficient matrix for the linear inequality constraints (from the `linear_inequality_constraint_matrix` *specification in* **MethodIndControl***)*

- RealVector linearIneqLowerBnds

  lower bounds for the linear inequality constraints (from the `linear_inequality_lower_bounds` *specification in* **MethodIndControl***)*

- RealVector linearIneqUpperBnds

  upper bounds for the linear inequality constraints (from the `linear_inequality_upper_bounds` *specification in* **MethodIndControl***)*

- RealVector linearIneqScales

  scaling factors for the linear inequality constraints (from the `linear_inequality_scales` *specification in* **MethodIndControl***)*

- RealVector linearEqConstraintCoeffs

  coefficient matrix for the linear equality constraints (from the `linear_equality_constraint_matrix` *specification in* **MethodIndControl***)*

- RealVector linearEqTargets

targets for the linear equality constraints (from the `linear_equality_targets` *specification in* **MethodInd-Control***)*

- RealVector linearEqScales

  scaling factors for the linear equality constraints (from the `linear_equality_scales` *specification in* **MethodIndControl***)*

- String methodName

  the method selection: one of the optimizer, least squares, nond, dace, or parameter study methods

- String minMaxType

  the `optimization_type` *specification in* **MethodDOTDC**

- 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

  the `search_method` *specification for Newton and nonlinear interior-point methods in* **MethodOPTPPDC**

- Real gradientTolerance

  the `gradient_tolerance` *specification in* **MethodOPTPPDC**

- Real maxStep

  the `max_step` *specification in* **MethodOPTPPDC**

- String meritFn

  the `merit_function` *specification for nonlinear interior-point methods in* **MethodOPTPPDC**

- String centralPath

  the `central_path` *specification for nonlinear interior-point methods in* **MethodOPTPPDC**

- Real stepLenToBoundary

  the `steplength_to_boundary` *specification for nonlinear interior-point methods in* **MethodOPTPPDC**

- Real centeringParam

  the `centering_parameter` *specification for nonlinear interior-point methods in* **MethodOPTPPDC**

- int searchSchemeSize

  the `search_scheme_size` *specification for PDS methods in* **MethodOPTPPDC**

- String evalSynchronization

  the `synchronization` *setting for parallel pattern search methods in* **MethodCOLINYPS** *and* **MethodCOLINYAPPS**

- Real constraintPenalty

  the initial `constraint_penalty` *for COLINY methods in* **MethodCOLINYAPPS**, **MethodCOLINYDIR**, **MethodCOLINYPS**, **MethodCOLINYSW** *and* **MethodCOLINYEA**

- bool constantPenalty

  the `constant_penalty` *flag for COLINY methods in* **MethodCOLINYPS** *and* **MethodCOLINYSW**

- Real globalBalanceParam

  the `global_balance_parameter` *for the DIRECT method in* **MethodCOLINYDIR**

- Real localBalanceParam

  the `local_balance_parameter` *for the DIRECT method in* **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**

- String boxDivision

  the `division` *setting (*`major_dimension` *or* `all_dimensions`*) for the DIRECT method in* **MethodCOLINYDIR**

- bool mutationAdaptive

  the `non_adaptive` *specification for the coliny_ea method in* **MethodCOLINYEA**

- bool showMiscOptions

  the `show_misc_options` *specification in* **MethodCOLINYDC**

- StringArray miscOptions

  the `misc_options` *specification in* **MethodCOLINYDC**

- Real solnAccuracy

  the `solution_accuracy` *specification in* **MethodCOLINYDC**

- Real crossoverRate

  the `crossover_rate` *specification for EA methods in* **MethodCOLINYEA**

- Real mutationRate

  the `mutation_rate` *specification for EA methods in* **MethodCOLINYEA**

- Real mutationScale

  the `mutation_scale` *specification for EA methods in* **MethodCOLINYEA**

- Real mutationMinScale

  the `min_scale` *specification for mutation in EA methods in* **MethodCOLINYEA**

- Real initDelta

  the `initial_delta` *specification for APPS/COBYLA/PS/SW methods in* **MethodCOLINYAPPS**, **MethodCOLINYCOB**, **MethodCOLINYPS**, *and* **MethodCOLINYSW**

- Real threshDelta

  the `threshold_delta` *specification for APPS/COBYLA/PS/SW methods in* **MethodCOLINYAPPS**, **MethodCOLINYCOB**, **MethodCOLINYPS**, *and* **MethodCOLINYSW**

- Real contractFactor

  the `contraction_factor` *specification for APPS/PS/SW methods in* **MethodCOLINYAPPS**, **MethodCOLINYPS**, *and* **MethodCOLINYSW**

- int newSolnsGenerated

  the `new_solutions_generated` *specification for GA/EPSA methods in* **MethodCOLINYEA**

- int numberRetained

  the integer assignment to random, chc, or elitist in the `replacement_type` *specification for GA/EPSA methods in* **MethodCOLINYEA**

- bool expansionFlag

  the `no_expansion` *specification for APPS/PS/SW methods in* **MethodCOLINYAPPS**, **MethodCOLINYPS**, *and* **MethodCOLINYSW**

- int expandAfterSuccess

  the `expand_after_success` *specification for PS/SW methods in* **MethodCOLINYPS** *and* **MethodCOLINYSW**

- int contractAfterFail

  the `contract_after_failure` *specification for the SW method in* **MethodCOLINYSW**

- int mutationRange

  the `mutation_range` *specification for the pga_int method in* **MethodCOLINYEA**

- int totalPatternSize

  the `total_pattern_size` *specification for PS methods in* **MethodCOLINYPS**

- bool randomizeOrderFlag

  the `stochastic` *specification for the PS method in* **MethodCOLINYPS**

- String selectionPressure

  the `fitness_type` *specification for EA methods in* **MethodCOLINYEA**

- String replacementType

  the `replacement_type` *specification for EA methods in* **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

  the `exploratory_moves` *specification for the PS method in* **MethodCOLINYPS**

- String patternBasis

  the `pattern_basis` *specification for APPS/PS methods in* **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

  The minimum percent change before convergence for a fitness tracker converger.

- size_t numGenerations

  The number of generations over which a fitness tracker converger should track.

- Real fitnessLimit

  The cutoff value for survival in fitness limiting selectors (e.g., below_limit selector).

- Real shrinkagePercent

  The minimum percentage of the requested number of selections that must take place on each call to the selector (0, 1).

- String nichingType

  The niching type.

- RealVector nicheVector

  The discretization percentage along each objective.

- String initializationType

  The means by which the JEGA should initialize the population.

- String flatFile

  The filename to use for initialization.

- String logFile

  The filename to use for logging.

- int populationSize

  the `population_size` *specification for GA methods in* **MethodCOLINYEA**

- bool printPopFlag

  The `print_each_pop` *flag to set the printing of the population at each generation.*

- String daceMethod

  the dace method selection: grid, random, oas, lhs, oa_lhs, box_behnken, or central_composite (from the `dace` *specification in* **MethodDDACE***)*

- int numSymbols

  the `symbols` *specification for DACE methods*

- bool mainEffectsFlag

  the `main_effects` *specification for sampling methods in* **MethodDDACE***)*

- bool latinizeFlag

  the `latinize` *specification for FSU QMC and CVT methods in* **MethodFSUDACE**

- bool volQualityFlag

  the `quality_metrics` *specification for sampling methods (FSU QMC and CVT methods in* **MethodFSUDACE***)*

- bool varBasedDecompFlag

  the `var_based_decomp` *specification for sampling methods (FSU QMC and CVT methods in* **MethodFSUDACE***)*

- IntVector sequenceStart

  the `sequenceStart` *specification in* **MethodFSUDACE**

- IntVector sequenceLeap

  the `sequenceLeap` *specification in* **MethodFSUDACE**

- IntVector primeBase

  the `primeBase` *specification in* **MethodFSUDACE**

- int numTrials

  the `numTrials` *specification in* **MethodFSUDACE**

- String trialType

  the `trial_type` *specification in* **MethodFSUDACE**

- int randomSeed

  the `seed` *specification for COLINY, NonD, & DACE methods*

- int numSamples

  the `samples` *specification for NonD & DACE methods*

- bool fixedSeedFlag

  flag for fixing the value of the seed among different NonD/DACE sample sets. This results in the use of the same sampling stencil/pattern throughout a strategy with repeated sampling.

- bool fixedSequenceFlag

  flag for fixing the sequence for Halton or Hammersley QMC sample sets. This results in the use of the same sampling stencil/pattern throughout a strategy with repeated sampling.

- int expansionTerms

the `expansion_terms` *specification in* **MethodNonDPCE**

- int expansionOrder

  the `expansion_order` *specification in* **MethodNonDPCE**

- String sampleType

  the `sample_type` *specification in* **MethodNonDMC** *and* **MethodNonDPCE**

- String reliabilitySearchType

  the type of MPP search as specified by `x_taylor_mean`, `x_taylor_mpp`, `u_taylor_mean`, `u_taylor_-mpp`, *or* `no_approx` *in* **MethodNonDRel**

- String reliabilitySearchAlgorithm

  the algorithm selection used for computing the MPP as specified by `sqp` *or* `nip` *in* **MethodNonDRel**

- String reliabilityIntegration

  the `first_order/second_order integration` *selection in* **MethodNonDRel**

- String distributionType

  the `distribution cumulative` *or* `complementary` *specification in* **MethodNonDMC**, **MethodNon-DPCE**, *and* **MethodNonDRel**

- String responseLevelMappingType

  the `compute probabilities` *or* `reliabilities` *specification in* **MethodNonDMC**, **MethodNonDPCE**, *and* **MethodNonDRel**

- RealVectorArray responseLevels

  the `response_levels` *specification in* **MethodNonDMC**, **MethodNonDPCE**, *and* **MethodNonDRel**

- RealVectorArray probabilityLevels

  the `probability_levels` *specification in* **MethodNonDMC**, **MethodNonDPCE**, *and* **MethodNonDRel**

- RealVectorArray reliabilityLevels

  the `reliability_levels` *specification in* **MethodNonDMC**, **MethodNonDPCE**, *and* **MethodNonDRel**

- bool allVarsFlag

  the `all_variables` *specification in* **MethodNonDMC**

- int paramStudyType

  the type of parameter study: list(-1), vector(1, 2, or 3), 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

  the `partitions` *specification for PStudy method in* **MethodPSMPS**

Private Member Functions

- void assign (const DataMethod &data_method)

  convenience function for setting this objects attributes equal to the attributes of the incoming data_method object (used by copy constructor and assignment operator)

10.25.1 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

10.26 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

  string identifier for the model specification data set (from the `id_model` *specification in* **ModelIndControl***)*

- String modelType

  model type selection: single, surrogate, or nested (from the model type specification in **ModelIndControl***)*

- String variablesPointer

  string pointer to the variables specification to be used by this model (from the `variables_pointer` *specification in* **ModelIndControl***)*

- String interfacePointer

  string pointer to the interface specification to be used by this model (from the `interface_pointer` *specification in* **ModelSingle** *and the* `optional_interface_pointer` *specification in* **ModelNested***)*

- String responsesPointer

  string pointer to the responses specification to be used by this model (from the `responses_pointer` *specification in* **ModelIndControl***)*

- IntArray surrogateFnIds

  array specifying the response function set that is approximated

- String approxType

  the selected approximation type: local_taylor, multipoint_tana, global_(neural_-network,mars,hermite,gaussian,polynomial,kriging), or hierarchical

- String actualModelPtr

  pointer to the interface specification for constructing the truth model used in building local and multipoint approximations (from the `actual_model_pointer` *specification in* **ModelSurrL** *and* **ModelSurrMP***)*

- String lowFidelityModelPtr

  pointer to the low fidelity model specification used in hierarchical approximations (from the `low_fidelity_-model_pointer` *specification in* **ModelSurrH***)*

- String highFidelityModelPtr

  pointer to the high fidelity model specification used in hierarchical approximations (from the `high_fidelity_-model_pointer` *specification in* **ModelSurrH***)*

- String approxDaceMethodPtr

  pointer to the design of experiments method used in building global approximations (from the `dace_method_-pointer` *specification in* **ModelSurrG***)*

- String approxSampleReuse

  sample reuse selection for building global approximations: none, all, region, or file (from the `reuse_samples` *specification in* **ModelSurrG***)*

- String approxSampleReuseFile

  the file name for the "file" setting for the `reuse_samples` *specification in* **ModelSurrG**

- String approxCorrectionType

  correction type for global and hierarchical approximations: additive or multiplicative (from the `correction` *specification in* **ModelSurrG** *and* **ModelSurrH***)*

- short approxCorrectionOrder

  correction order for global and hierarchical approximations: 0, 1, or 2 (from the `correction` *specification in* **ModelSurrG** *and* **ModelSurrH***)*

- bool approxGradUsageFlag

  flags the use of gradients in building global approximations (from the `use_gradients` *specification in* **ModelSurrG***)*

- RealVector krigingCorrelations

  vector of correlations used in building a kriging approximation (from the `correlations` *specification in* **ModelSurrG***)*

- short polynomialOrder

  scalar integer indicating the order of the polynomial approximation (1=linear, 2=quadratic, 3=cubic; from the `polynomial` *specification in* **ModelSurrG***)*

- String optionalInterfRespPointer

  string pointer to the responses specification used by the optional interface in nested models (from the `optional_interface_responses_pointer` *specification in* **ModelNested***)*

- String subMethodPointer

  string pointer to the sub-iterator used by nested models (from the `sub_method_pointer` *specification in* **ModelNested***)*

- StringArray primaryVarMaps

  the primary variable mappings used in nested models for identifying the lower level variable targets for inserting top level variable values (from the `primary_variable_mapping` *specification in* **ModelNested***)*

- StringArray secondaryVarMaps

  the secondary variable mappings used in nested models for identifying the (distribution) parameter targets within the lower level variables for inserting top level variable values (from the `secondary_variable_mapping` *specification in* **ModelNested***)*

- RealVector primaryRespCoeffs

  the primary response mapping matrix used in nested models for weighting contributions from the sub-iterator responses in the top level (objective) functions (from the `primary_response_mapping` *specification in* **ModelNested***)*

- RealVector secondaryRespCoeffs

  the secondary response mapping matrix used in nested models for weighting contributions from the sub-iterator responses in the top level (constraint) functions (from the `secondary_response_mapping` *specification in* **ModelNested***)*

Private Member Functions

- void assign (const DataModel &data_model)

  convenience function for setting this objects attributes equal to the attributes of the incoming data_model object (used by copy constructor and assignment operator)

10.26.1 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

10.27 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

  number of objective functions (from the `num_objective_functions` *specification in* **RespFnOpt***)*

- size_t numNonlinearIneqConstraints

  number of nonlinear inequality constraints (from the `num_nonlinear_inequality_constraints` *specification in* **RespFnOpt***)*

- size_t numNonlinearEqConstraints

number of nonlinear equality constraints (from the `num_nonlinear_equality_constraints` *specification in* **RespFnOpt***)*

- size_t numLeastSqTerms

  number of least squares terms (from the `num_least_squares_terms` *specification in* **RespFnLS***)*

- size_t numResponseFunctions

  number of generic response functions (from the `num_response_functions` *specification in* **RespFnGen***)*

- RealVector objectiveFunctionScales

  vector of objective function scaling factors (from the `objective_function_scales` *specification in* **RespFnOpt***)*

- RealVector multiObjectiveWeights

  vector of multiobjective weightings (from the `multi_objective_weights` *specification in* **RespFnOpt***)*

- RealVector leastSqTermScales

  vector of least squares term scaling factors (from the `least_squares_term_scales` *specification in* **RespFnOpt***)*

- RealVector nonlinearIneqLowerBnds

  vector of nonlinear inequality constraint lower bounds (from the `nonlinear_inequality_lower_bounds` *specification in* **RespFnOpt***)*

- RealVector nonlinearIneqUpperBnds

  vector of nonlinear inequality constraint upper bounds (from the `nonlinear_inequality_upper_bounds` *specification in* **RespFnOpt***)*

- RealVector nonlinearIneqScales

  vector of nonlinear inequality constraint scaling factors (from the `nonlinear_inequality_scales` *specification in* **RespFnOpt***)*

- RealVector nonlinearEqTargets

  vector of nonlinear equality constraint targets (from the `nonlinear_equality_targets` *specification in* **RespFnOpt***)*

- RealVector nonlinearEqScales

  vector of nonlinear equality constraint scaling factors (from the `nonlinear_equality_scales` *specification in* **RespFnOpt***)*

- String gradientType

  gradient type: none, numerical, analytic, or mixed (from the `no_gradients`, `numerical_gradients`, `analytic_gradients`*, and* `mixed_gradients` *specifications in* **RespGrad***)*

- String hessianType

  Hessian type: none, numerical, quasi, analytic, or mixed (from the `no_hessians`, `numerical_hessians`, `quasi_hessians`, `analytic_hessians`*, and* `mixed_hessians` *specifications in* **RespHess***).*

- String quasiHessianType

  quasi-Hessian type: bfgs, damped_bfgs, or sr1 (from the `bfgs` *and* `sr1` *specifications in* **RespHess***)*

- String methodSource

  numerical gradient method source: dakota or vendor (from the `method_source` *specification in* **RespGradNum** *and* **RespGradMixed***)*

- String intervalType

  numerical gradient interval type: forward or central (from the `interval_type` *specification in* **RespGradNum** *and* **RespGradMixed***)*

- RealVector fdGradStepSize

  vector of finite difference step sizes for numerical gradients, one step size per active continuous variable, used in computing 1st-order forward or central differences (from the `fd_gradient_step_size` *specification in* **RespGradNum** *and* **RespGradMixed***)*

- RealVector fdHessStepSize

  vector of finite difference step sizes for numerical Hessians, one step size per active continuous variable, used in computing 1st-order gradient-based differences and 2nd-order function-based differences (from the `fd_-hessian_step_size` *specification in* **RespHessNum** *and* **RespHessMixed***)*

- IntList idNumericalGrads

  mixed gradient numerical identifiers (from the `id_numerical_gradients` *specification in* **RespGradMixed***)*

- IntList idAnalyticGrads

  mixed gradient analytic identifiers (from the `id_analytic_gradients` *specification in* **RespGradMixed***)*

- IntList idNumericalHessians

  mixed Hessian numerical identifiers (from the `id_numerical_hessians` *specification in* **RespHessMixed***)*

- IntList idQuasiHessians

  mixed Hessian quasi identifiers (from the `id_quasi_hessians` *specification in* **RespHessMixed***)*

- IntList idAnalyticHessians

  mixed Hessian analytic identifiers (from the `id_analytic_hessians` *specification in* **RespHessMixed***)*

- String idResponses

  string identifier for the responses specification data set (from the `id_responses` *specification in* **RespSetId***)*

- StringArray responseLabels

  the response labels array (from the `response_descriptors` *specification in* **RespLabels***)*

Private Member Functions

- void assign (const DataResponses &data_responses)

convenience function for setting this objects attributes equal to the attributes of the incoming data_responses object (used by copy constructor and assignment operator)

10.27.1 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 specification 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::responsesList 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

10.28 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

  the strategy selection: multi_level, surrogate_based_opt, branch_and_bound, multi_start, pareto_set, or single_-method

- bool graphicsFlag

  flags use of graphics by the strategy (from the graphics specification in **StratIndControl***)*

- bool tabularDataFlag

  flags tabular data collection by the strategy (from the tabular_graphics_data specification in **StratIndControl***)*

- String tabularDataFile

the filename used for tabular data collection by the strategy (from the tabular_graphics_file specification in **StratIndControl***)*

- int iteratorServers

  number of servers for concurrent iterator parallelism (from the iterator_servers specification in **StratIndControl***)*

- String iteratorScheduling

  type of scheduling (self or static) used in concurrent iterator parallelism (from the iterator_self_scheduling and iterator_static_scheduling specifications in **StratIndControl***)*

- String methodPointer

  method identifier for the strategy (from the opt_method_pointer specifications in **StratSBO** *and* **StratParetoSet** *and method_pointer specifications in* **StratSingle** *and* **StratMultiStart***)*

- StringArray multilevelMethodList

  array of methods for the multilevel hybrid optimization strategy (from the method_list specification in **StratML***)*

- String multilevelType

  the type of multilevel hybrid optimization strategy: uncoupled, uncoupled_adaptive, or coupled (from the uncoupled, adaptive, and coupled specifications in **StratML***)*

- Real multilevelProgThresh

  progress threshold for uncoupled_adaptive multilevel hybrids (from the progress_threshold specification in **StratML***)*

- String multilevelGlobalMethodPointer

  global method pointer for coupled multilevel hybrids (from the global_method_pointer specification in **StratML***)*

- String multilevelLocalMethodPointer

  local method pointer for coupled multilevel hybrids (from the local_method_pointer specification in **StratML***)*

- Real multilevelLSProb

  local search probability for coupled multilevel hybrids (from the local_search_probability specification in **StratML***)*

- int surrBasedOptMaxIterations

  maximum number of iterations in the surrogate-based optimization strategy (from the max_iterations specification in **StratSBO***)*

- Real surrBasedOptConvTol

  convergence tolerance in the surrogate-based optimization strategy (from the convergence_tolerance specification in **StratSBO***)*

- int surrBasedOptSoftConvLimit

  number of consecutive iterations with change less than surrBasedOptConvTol required to trigger convergence within the surrogate-based optimization strategy (from the soft_convergence_limit specification in **StratSBO***)*

- bool surrBasedOptLayerBypass

  flag to indicate user-specification of a bypass of any/all layerings in evaluating truth response values in SBO.

- Real surrBasedOptTRInitSize

  initial trust region size in the surrogate-based optimization strategy (from the initial_size specification in **StratSBO***) note: this is a relative value, e.g., 0.1 = 10% of global bounds distance (upper bound - lower bound) for each variable*

- Real surrBasedOptTRMinSize

  minimum trust region size in the surrogate-based optimization strategy (from the minimum_size specification in **StratSBO***), if the trust region size falls below this threshold the SBO iterations are terminated (note: if kriging is used with SBO, the min trust region size is set to 1.0e-3 in attempt to avoid ill-conditioned matrixes that arise in kriging over small trust regions)*

- Real surrBasedOptTRContractTrigger

  trust region minimum improvement level (ratio of actual to predicted decrease in objective fcn) in the surrogate-based optimization strategy (from the `contract_threshold` *specification in* **StratSBO***), the trust region shrinks or is rejected if the ratio is below this value ("eta_1" in the Conn-Gould-Toint trust region book)*

- Real surrBasedOptTRExpandTrigger

  trust region sufficient improvement level (ratio of actual to predicted decrease in objective fn) in the surrogate-based optimization strategy (from the `expand_threshold` *specification in* **StratSBO***), the trust region expands if the ratio is above this value ("eta_2" in the Conn-Gould-Toint trust region book)*

- Real surrBasedOptTRContract

  trust region contraction factor in the surrogate-based optimization strategy (from the contraction_factor specification in **StratSBO***)*

- Real surrBasedOptTRExpand

  trust region expansion factor in the surrogate-based optimization strategy (from the expansion_factor specification in **StratSBO***)*

- bool surrBasedOptTRConstraintRelax

  flag to use trust region constraint relaxation for infeasible points

- int surrBasedOptTRConstraintRelaxMethod

  trust region constraint relaxation method (currently implements: homotopy)

- int concurrentRandomJobs

  number of random jobs to perform in the concurrent strategy (from the random_starts and random_weight_sets specifications in **StratMultiStart** *and* **StratParetoSet***)*

- int concurrentSeed

  seed for the selected random jobs within the concurrent strategy (from the seed specification in **StratMultiStart** *and* **StratParetoSet***)*

- RealVector concurrentParameterSets

  user-specified (i.e., nonrandom) parameter sets to evaluate in the concurrent strategy (from the starting_points and multi_objective_weight_sets specifications in **StratMultiStart** *and* **StratParetoSet***)*

Private Member Functions

- void assign (const DataStrategy &data_strategy)

  convenience function for setting this objects attributes equal to the attributes of the incoming data_strategy object (used by copy constructor and assignment operator)

10.28.1 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. Default 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

10.29 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

- size_t num_discrete_variables ()

  return total number of discrete variables

- size_t num_variables ()

  return total number of variables

Public Attributes

- String idVariables

  string identifier for the variables specification data set (from the `id_variables` *specification in* **VarSetId***)*

- size_t numContinuousDesVars

  number of continuous design variables (from the `continuous_design` *specification in* **VarDV***)*

- size_t numDiscreteDesVars

  number of discrete design variables (from the `discrete_design` *specification in* **VarDV***)*

- size_t numNormalUncVars

  number of normal uncertain variables (from the `normal_uncertain` *specification in* **VarUV***)*

- size_t numLognormalUncVars

  number of lognormal uncertain variables (from the `lognormal_uncertain` *specification in* **VarUV***)*

- size_t numUniformUncVars

  number of uniform uncertain variables (from the `uniform_uncertain` *specification in* **VarUV***)*

- size_t numLoguniformUncVars

  number of loguniform uncertain variables (from the `loguniform_uncertain` *specification in* **VarUV***)*

- size_t numTriangularUncVars

  number of triangular uncertain variables (from the `triangular_uncertain` *specification in* **VarUV***)*

- size_t numWeibullUncVars

  number of weibull uncertain variables (from the `weibull_uncertain` *specification in* **VarUV***)*

- size_t numBetaUncVars

  number of beta uncertain variables (from the `beta_uncertain` *specification in* **VarUV***)*

- size_t numGammaUncVars

  number of gamma uncertain variables (from the `gamma_uncertain` *specification in* **VarUV***)*

- size_t numFrechetUncVars

  number of frechet uncertain variables (from the `frechet_uncertain` *specification in* **VarUV***)*

- size_t numGumbelUncVars

number of gumbel uncertain variables (from the `gumbel_uncertain` *specification in* **VarUV***)*

- size_t numHistogramUncVars

  number of histogram uncertain variables (from the `histogram_uncertain` *specification in* **VarUV***)*

- size_t numIntervalUncVars

  number of interval uncertain variables (from the `interval_uncertain` *specification in* **VarUV***)*

- size_t numContinuousStateVars

  number of continuous state variables (from the `continuous_state` *specification in* **VarSV***)*

- size_t numDiscreteStateVars

  number of discrete state variables (from the `discrete_state` *specification in* **VarSV***)*

- RealVector continuousDesignVars

  initial values for the continuous design variables array (from the `cdv_initial_point` *specification in* **VarDV***)*

- RealVector continuousDesignLowerBnds

  the continuous design lower bounds array (from the `cdv_lower_bounds` *specification in* **VarDV***)*

- RealVector continuousDesignUpperBnds

  the continuous design upper bounds array (from the `cdv_upper_bounds` *specification in* **VarDV***)*

- RealVector continuousDesignScales

  the continuous design scales array (from the `cdv_scales` *specification in* **VarDV***)*

- IntVector discreteDesignVars

  initial values for the discrete design variables array (from the `ddv_initial_point` *specification in* **VarDV***)*

- IntVector discreteDesignLowerBnds

  the discrete design lower bounds array (from the `ddv_lower_bounds` *specification in* **VarDV***)*

- IntVector discreteDesignUpperBnds

  the discrete design upper bounds array (from the `ddv_upper_bounds` *specification in* **VarDV***)*

- StringArray continuousDesignLabels

  the continuous design labels array (from the `cdv_descriptors` *specification in* **VarDV***)*

- StringArray discreteDesignLabels

  the discrete design labels array (from the `ddv_descriptors` *specification in* **VarDV***)*

- RealVector normalUncMeans

  means of the normal uncertain variables (from the `nuv_means` *specification in* **VarUV***)*

- RealVector normalUncStdDevs

  standard deviations of the normal uncertain variables (from the `nuv_std_deviations` *specification in* **VarUV***)*

- RealVector normalUncLowerBnds

  distribution lower bounds for the normal uncertain variables (from the `nuv_lower_bounds` *specification in* **VarUV***)*

- RealVector normalUncUpperBnds

  distribution upper bounds for the normal uncertain variables (from the `nuv_upper_bounds` *specification in* **VarUV***)*

- RealVector lognormalUncMeans

  means of the lognormal uncertain variables (from the `lnuv_means` *specification in* **VarUV***)*

- RealVector lognormalUncStdDevs

  standard deviations of the lognormal uncertain variables (from the `lnuv_std_deviations` *specification in* **VarUV***)*

- RealVector lognormalUncErrFacts

  error factors for the lognormal uncertain variables (from the `lnuv_error_factors` *specification in* **VarUV***)*

- RealVector lognormalUncLowerBnds

  distribution lower bounds for the lognormal uncertain variables (from the `lnuv_lower_bounds` *specification in* **VarUV***)*

- RealVector lognormalUncUpperBnds

  distribution upper bounds for the lognormal uncertain variables (from the `lnuv_upper_bounds` *specification in* **VarUV***)*

- RealVector uniformUncLowerBnds

  distribution lower bounds for the uniform uncertain variables (from the `uuv_lower_bounds` *specification in* **VarUV***)*

- RealVector uniformUncUpperBnds

  distribution upper bounds for the uniform uncertain variables (from the `uuv_upper_bounds` *specification in* **VarUV***)*

- RealVector loguniformUncLowerBnds

  distribution lower bounds for the loguniform uncertain variables (from the `luuv_lower_bounds` *specification in* **VarUV***)*

- RealVector loguniformUncUpperBnds

  distribution upper bounds for the loguniform uncertain variables (from the `luuv_upper_bounds` *specification in* **VarUV***)*

- RealVector triangularUncModes

  modes of the triangular uncertain variables (from the `tuv_modes` *specification in* **VarUV***)*

- RealVector triangularUncLowerBnds

distribution lower bounds for the triangular uncertain variables (from the `tuv_lower_bounds` *specification in* **VarUV***)*

- RealVector triangularUncUpperBnds

  distribution upper bounds for the triangular uncertain variables (from the `tuv_upper_bounds` *specification in* **VarUV***)*

- RealVector betaUncAlphas

  alpha factors for the beta uncertain variables (from the `buv_means` *specification in* **VarUV***)*

- RealVector betaUncBetas

  beta factors for the beta uncertain variables (from the `buv_std_deviations` *specification in* **VarUV***)*

- RealVector betaUncLowerBnds

  distribution lower bounds for the beta uncertain variables (from the `buv_lower_bounds` *specification in* **VarUV***)*

- RealVector betaUncUpperBnds

  distribution upper bounds for the beta uncertain variables (from the `buv_upper_bounds` *specification in* **VarUV***)*

- RealVector gammaUncAlphas

  alpha factors for the gamma uncertain variables (from the `gauv_alphas` *specification in* **VarUV***)*

- RealVector gammaUncBetas

  beta factors for the gamma uncertain variables (from the `gauv_betas` *specification in* **VarUV***)*

- RealVector gumbelUncAlphas

  alpha factors for the gumbel uncertain variables (from the `guuv_alphas` *specification in* **VarUV***)*

- RealVector gumbelUncBetas

  beta factors for of the gumbel uncertain variables (from the `guuv_betas` *specification in* **VarUV***)*

- RealVector frechetUncAlphas

  alpha factors for the frechet uncertain variables (from the `fuv_alphas` *specification in* **VarUV***)*

- RealVector frechetUncBetas

  beta factors for the frechet uncertain variables (from the `fuv_betas` *specification in* **VarUV***)*

- RealVector weibullUncAlphas

  alpha factors for the weibull uncertain variables (from the `wuv_alphas` *specification in* **VarUV***)*

- RealVector weibullUncBetas

  beta factors for the weibull uncertain variables (from the `wuv_betas` *specification in* **VarUV***)*

- RealVectorArray histogramUncBinPairs

an array containing a vector of (x,y) pairs for each bin-based histogram uncertain variable (see continuous linear histogram in LHS manual; from the `huv_num_bin_pairs` *and* `huv_bin_pairs` *specifications in* **VarUV***)*

- RealVectorArray histogramUncPointPairs

  an array containing a vector of (x,y) pairs for each point-based histogram uncertain variable (see discrete histogram in LHS manual; from the `huv_num_point_pairs` *and* `huv_point_pairs` *specifications in* **VarUV***)*

- IntVector intervalUncNumIntervals

  number of intervals per interval uncertain variables (from the `iuv_num_intervals` *specification in* **VarUV***)*

- RealVector intervalUncProbValues

  Probability values per interval uncertain variables (from the `iuv_interval_probs` *specification in* **VarUV***).*

- RealVector intervalUncIntervalBounds

  Interval Bounds per interval uncertain variables (from the `iuv_interval_bounds` *specification in* **VarUV***).*

- RealMatrix uncertainCorrelations

  correlation matrix for all uncertain variables (from the `uncertain_correlation_matrix` *specification in* **VarUV***). This matrix specifies rank correlations for sampling methods (i.e., LHS) and correlation coefficients (rho_-ij = normalized covariance matrix) for analytic reliability methods.*

- RealVector uncertainVars

  array of values for all uncertain variables (built and initialized in ProblemDescDB::variables_kwhandler())

- RealVector uncertainLowerBnds

  distribution lower bounds for all uncertain variables (collected from `nuv_lower_bounds`, `lnuv_lower_-bounds`, `uuv_lower_bounds`, `luuv_lower_bounds`, `tuv_lower_bounds`, *and* `buv_lower_-bounds` *specifications in* **VarUV***, and derived for gamma, gumbel, frechet, weibull and histogram specifications)*

- RealVector uncertainUpperBnds

  distribution upper bounds for all uncertain variables (collected from `nuv_upper_bounds`, `lnuv_upper_-bounds`, `uuv_upper_bounds`, `luuv_upper_bounds`, `tuv_lower_bounds`, *and* `buv_upper_-bounds` *specifications in* **VarUV***, and derived for gamma, gumbel, frechet, weibull and histogram specifications)*

- StringArray uncertainLabels

  labels for all uncertain variables (collected from `nuv_descriptors`, `lnuv_descriptors`, `uuv_-descriptors`, `luuv_descriptors`, `tuv_descriptors`, `buv_descriptors`, `gauv_-descriptors`, `guuv_descriptors`, `fuv_descriptors`, `wuv_descriptors`, *and* `huv_-descriptors` *specifications in* **VarUV***)*

- RealVector continuousStateVars

  initial values for the continuous state variables array (from the `csv_initial_state` *specification in* **VarSV***)*

- RealVector continuousStateLowerBnds

  the continuous state lower bounds array (from the `csv_lower_bounds` *specification in* **VarSV***)*

- RealVector continuousStateUpperBnds

  the continuous state upper bounds array (from the `csv_upper_bounds` *specification in* **VarSV***)*

- IntVector discreteStateVars

  initial values for the discrete state variables array (from the `dsv_initial_state` *specification in* **VarSV***)*

- IntVector discreteStateLowerBnds

  the discrete state lower bounds array (from the `dsv_lower_bounds` *specification in* **VarSV***)*

- IntVector discreteStateUpperBnds

  the discrete state upper bounds array (from the `dsv_upper_bounds` *specification in* **VarSV***)*

- StringArray continuousStateLabels

  the continuous state labels array (from the `csv_descriptors` *specification in* **VarSV***)*

- StringArray discreteStateLabels

  the discrete state labels array (from the `dsv_descriptors` *specification in* **VarSV***)*

Private Member Functions

- void assign (const DataVariables &data_variables)

  convenience function for setting this objects attributes equal to the attributes of the incoming data_variables object (used by copy constructor and assignment operator)

10.29.1 Detailed Description

Container class for variables specification data.

The DataVariables class is used to contain the data from a variables keyword specification. It is populated by ProblemDescDB::variables_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataVariables objects is maintained in ProblemDescDB::variablesList, one for each variables specification in an input file. Default values are managed in the DataVariables constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::variablesList is private (a similar model is used with SurrogateDataPoint objects contained in Dakota::Approximation).

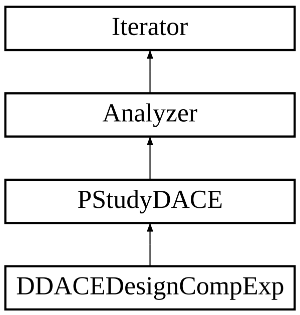
The documentation for this class was generated from the following files:

- DataVariables.H
- DataVariables.C

10.30 DDACEDesignCompExp Class Reference

Wrapper class for the DDACE design of experiments library.

Inheritance diagram for DDACEDesignCompExp::



Public Member Functions

- DDACEDesignCompExp (Model &model)

    primary constructor for building a standard DACE iterator

- ∼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 ()

    Returns one block of samples (ndim ∗ num_samples).

Private Member Functions

- void compute_main_effects ()

  builds a DDaceMainEffects::OneWayANOVA if mainEffectsFlag is set

- void resolve_samples_symbols ()

  convenience function for resolving number of samples and number of symbols from input.

Private Attributes

- String daceMethod

  oas, lhs, oa_lhs, random, box_behnken, central_composite, or grid

- int samplesSpec

  user specification of number of samples

- int numSamples

  number of samples to be evaluated

- int numSymbols

  number of symbols to be used in generating the sample set (inversely related to number of replications)

- const int originalSeed

  the user seed specification for the random number generator (allows repeatable results)

- int randomSeed

  current seed for the random number generator

- bool allDataFlag

  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()

- size_t numDACERuns

  counter for number of run() executions for this object

- bool varyPattern

  flag for continuing the random number sequence from a previous run() execution (e.g., for surrogate-based optimization) so that multiple executions are repeatable but not correlated.

- bool volQualityFlag

  flag which specifies evaluating the volumetric quality measures

- bool varBasedDecompFlag

  flag which specifies variance based decomposition

- bool mainEffectsFlag

  flag which specifies main effects

- std::vector< std::vector< int > > symbolMapping

  mapping of symbols for main effects calculations

10.30.1 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.

10.30.2 Constructor & Destructor Documentation

10.30.2.1 DDACEDesignCompExp (Model & *model*)

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

10.30.3 Member Function Documentation

10.30.3.1 void resolve_samples_symbols () `[private]`

convenience function for resolving number of samples and number of symbols from input.

This function must define a combination of samples and symbols that is acceptable for a particular sampling algorithm. Users provide requests for these quantities, but this function must enforce any restrictions imposed by the sampling algorithms.

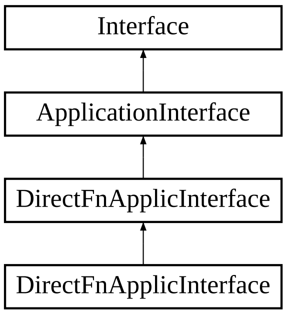
The documentation for this class was generated from the following files:

- DDACEDesignCompExp.H
- DDACEDesignCompExp.C

10.31 DirectFnApplicInterface Class Reference

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

Inheritance diagram for DirectFnApplicInterface::



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)

  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.

- void derived_map_asynch (const ParamResponsePair &pair)

  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.

- void derived_synch (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- void derived_synch_nowait (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- int 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)

  convenience function for local test simulators which sets variable attributes and zeros response data

- void overlay_response (Response &response)

  convenience function for local test simulators which overlays response contributions from multiple analyses using MPI_Reduce

Protected Attributes

- String iFilterName

  name of the direct function input filter

- String oFilterName

  name of the direct function output filter

- 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 discete 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

- IntArray 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

  the set of analyses within each function evaluation (from the analysis_drivers interface specification)

- size_t analysisDriverIndex

  the index of the active analysis driver within analysisDrivers

- String2DArray analysisComponents

  the set of optional analysis components used by the analysis drivers (from the analysis_components interface specification)

- 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 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 log_ratio ()

    the log_ratio UQ test function

- int short_column ()

    the short_column UQ/OUU test function

- 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

10.31.1 Detailed Description

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

DirectFnApplicInterface uses a few linkable simulation codes and several internal member functions to perform parameter to response mappings.

10.31.2 Member Function Documentation

10.31.2.1 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.

10.31.2.2 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);
```

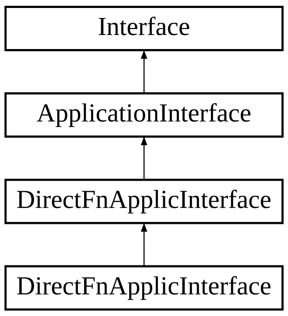
The documentation for this class was generated from the following files:

- DirectFnApplicInterface.H
- DirectFnApplicInterface.C

10.32 DirectFnApplicInterface Class Reference

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

Inheritance diagram for 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

10.32.1 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 uncommenting the LIBRARY_MODE_DEBUG define in main.C (which activates the plug-in code block within that file) and uncommenting the PLUGIN_S/PLUGIN_O declarations at the top of the Dakota/src Makefile (which add this class to the build). Test input files should then use an analysis_driver of "plugin_rosenbrock".

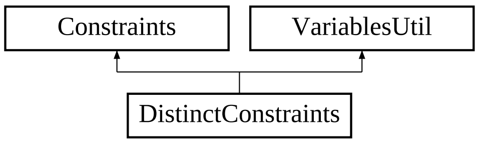
The documentation for this class was generated from the following files:

- PluginDirectFnApplicInterface.H
- PluginDirectFnApplicInterface.C

10.33 DistinctConstraints Class Reference

Derived class within the Constraints hierarchy which employs the default data view (no variable or domain type array merging).

Inheritance diagram for 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

- RealVector all_continuous_upper_bounds () const

  returns a single array with all continuous upper bounds

- IntVector all_discrete_lower_bounds () const

  returns a single array with all discrete lower bounds

- IntVector all_discrete_upper_bounds () const

  returns a single array with all discrete upper bounds

- 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

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

- RealVector continuousStateUpperBnds

  the continuous state upper bounds array

- IntVector discreteStateLowerBnds

  the discrete state lower bounds array

- IntVector discreteStateUpperBnds

  the discrete state upper bounds array

10.33.1 Detailed Description

Derived class within the Constraints hierarchy which employs 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).

10.33.2 Constructor & Destructor Documentation

10.33.2.1 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).

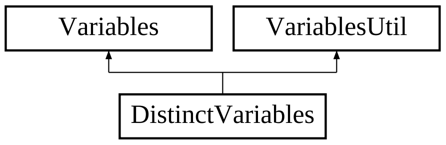
The documentation for this class was generated from the following files:

- DistinctConstraints.H
- DistinctConstraints.C

10.34 DistinctVariables Class Reference

Derived class within the Variables hierarchy which employs the default data view (no variable or domain type array merging).

Inheritance diagram for DistinctVariables::



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

- 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

- StringArray all_discrete_variable_labels () const

    returns a single array with all discrete variable labels

- 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

Private Member Functions

- void copy_rep (const Variables ∗vars_rep)

  Used by copy() to copy the contents of a letter class.

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

Friends

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

  equality operator

10.34.1 Detailed Description

Derived class within the Variables hierarchy which employs 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)).

10.34.2 Constructor & Destructor Documentation

10.34.2.1 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).

10.34.3 Friends And Related Function Documentation

10.34.3.1 bool operator== (const DistinctVariables & *vars1*, const DistinctVariables & *vars2*) `[friend]`

equality operator

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

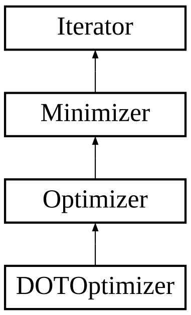
The documentation for this class was generated from the following files:

- DistinctVariables.H
- DistinctVariables.C

10.35 DOTOptimizer Class Reference

Wrapper class for the DOT optimization library.

Inheritance diagram for DOTOptimizer::



Public Member Functions

- DOTOptimizer (Model &model)

  constructor

- ~DOTOptimizer ()

  destructor

- void find_optimum ()

  Used within the optimizer branch for computing the optimal solution. Redefines the run_iterator virtual function for the optimizer branch.

Protected Member Functions

- virtual void derived_pre_run ()

  performs run-time set up

- virtual void derived_post_run ()

  performs final solution processing

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

- SizetList constraintMappingIndices

  a list of indices for referencing the corresponding Response constraints used in computing the DOT constraints.

- RealList constraintMappingMultipliers

  a list of multipliers for mapping the Response constraints to the DOT constraints.

- RealList constraintMappingOffsets

  a list of offsets for mapping the Response constraints to the DOT constraints.

10.35.1 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.

10.35.2 Member Data Documentation

10.35.2.1 int dotInfo `[private]`

INFO from DOT manual.

Information requested by DOT: 0=optimization complete, 1=get values, 2=get gradients

10.35.2.2 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).

10.35.2.3 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.

10.35.2.4 int printControl `[private]`

IPRINT from DOT manual (controls output verbosity).

Values range from 0 (least output) to 7 (most output).

10.35.2.5 int optimizationType `[private]`

MINMAX from DOT manual (minimize or maximize).

Values of 0 or -1 (minimize) or 1 (maximize).

10.35.2.6 RealArray realCntlParmArray `[private]`

RPRM from DOT manual.

Array of real control parameters.

10.35.2.7 IntArray intCntlParmArray `[private]`

IPRM from DOT manual.

Array of integer control parameters.

10.35.2.8 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).

10.35.2.9 SizetList constraintMappingIndices `[private]`

a list of indices for referencing the corresponding 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.

10.35.2.10 RealList constraintMappingMultipliers `[private]`

a list of multipliers for mapping the Response constraints to 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.

10.35.2.11 RealList constraintMappingOffsets `[private]`

a list of offsets for mapping the Response constraints to the 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

10.36 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.

10.36.1 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.

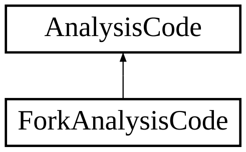
The documentation for this struct was generated from the following file:

- CtelRegExp.C

10.37 ForkAnalysisCode Class Reference

Derived class in the AnalysisCode class hierarchy which spawns simulations using forks.

Inheritance diagram for ForkAnalysisCode::



Public Member Functions

- ForkAnalysisCode (const ProblemDescDB &problem_db)

  constructor

- ~ForkAnalysisCode ()

  destructor

- pid_t fork_program (const bool block_flag)

  spawn a child process using fork()/vfork()/execvp() and wait for completion using waitpid() if block_flag is true

- void check_status (const int status)

  check the exit status of a forked process and abort if an 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

  an array of strings for use with execvp(const char ∗, char ∗ const ∗). These are converted to an array of const char∗'s in fork_program().

10.37.1 Detailed Description

Derived class in the AnalysisCode class hierarchy which spawns 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.

10.37.2 Member Function Documentation

10.37.2.1 void check_status (const int *status*)

check the exit status of a forked process and abort if an error code was returned

Check to see if the process terminated abnormally (WIFEXITED(status)==0) or if either execvp or the application returned a status code of -1 (WIFEXITED(status)!=0 && (signed char)WEXITSTATUS(status)==-1). If one of these conditions is detected, output a failure message and abort. Note: the application code should not return a status code of -1 unless an immediate abort of dakota is wanted. If for instance, failure capturing is to be used, the application code should write the word "FAIL" to the appropriate results file and return a status code of 0 through exit().

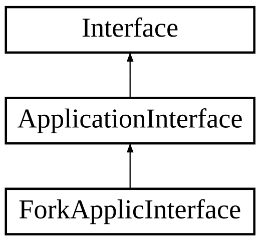
The documentation for this class was generated from the following files:

- ForkAnalysisCode.H
- ForkAnalysisCode.C

10.38 ForkApplicInterface Class Reference

Derived application interface class which spawns simulation codes using forks.

Inheritance diagram for 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)

  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.

- void derived_map_asynch (const ParamResponsePair &pair)

  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.

- void derived_synch (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- void derived_synch_nowait (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- int 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 derived_synch_kernel (PRPList &prp_list, const pid_t pid)

  Convenience function for common code between derived_synch() & derived_synch_nowait().

- pid_t fork_application (const bool block_flag)

  perform the complete function evaluation by managing the input 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

  ForkAnalysisCode provides convenience functions for forking individual programs and checking fork exit status.

- std::map< pid_t, int > processIdMap

  map of fork process id's to function evaluation id's for asynchronous evaluations

10.38.1 Detailed Description

Derived application interface class which spawns simulation codes using forks.

ForkApplicInterface uses a ForkAnalysisCode object for performing simulation invocations.

10.38.2 Member Function Documentation

10.38.2.1 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.

10.38.2.2 pid_t fork_application (const bool *block_flag*) [private]

perform the complete function evaluation by managing the input 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.

10.38.2.3 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 after ApplicationInterface::asynchronous_local_evaluations(). NOTE: This function should be elevated to ApplicationInterface if and when another derived interface class supports asynchronous local analyses.

10.38.2.4 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().

10.38.2.5 void serve_analyses_asynch () [private]

serve the analysis scheduler and execute analysis jobs asynchronously

This code runs multiple asynch analyses on each server. It is modeled after ApplicationInterface::serve_evaluations_asynch(). NOTE: This fn should be elevated to ApplicationInterface if and when another derived interface class supports hybrid analysis parallelism.

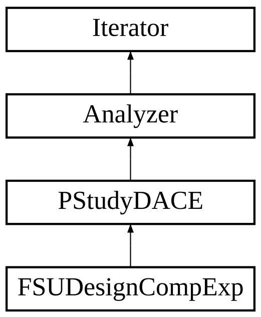
The documentation for this class was generated from the following files:

- ForkApplicInterface.H
- ForkApplicInterface.C

10.39 FSUDesignCompExp Class Reference

Wrapper class for the FSUDace QMC/CVT library.

Inheritance diagram for FSUDesignCompExp::



Public Member Functions

- FSUDesignCompExp (Model &model)

  primary constructor for building a standard DACE iterator

- ∼FSUDesignCompExp ()

  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 ()

  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

  user specification of number of samples

- int numSamples

  number of samples to be evaluated

- bool allDataFlag

  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()

- size_t numDACERuns

  counter for number of run() executions for this object

- bool latinizeFlag

  flag which specifies latinization of QMC or CVT sample sets

- bool volQualityFlag

  flag which specifies evaluating the volumetric quality measures

- bool varBasedDecompFlag

  flag which specifies calculating variance based decomposition sensitivity analysis metrics

- IntVector sequenceStart

  Integer vector defining a starting index into the sequence for random variable sampled. Default is 0 0 0 (e.g. for three random variables).

- IntVector sequenceLeap

  Integer vector defining the leap number for each sequence being generated. Default is 1 1 1 (e.g. for three random vars.).

- IntVector primeBase

  Integer vector defining the prime base for each sequence being generated. Default is 2 3 5 (e.g., for three random vars.).

- int originalSeed

  the user seed specification for the random number generator (allows repeatable results)

- int randomSeed

current seed for the random number generator

- bool varyPattern

  flag for continuing the random number or QMC sequence from a previous run() execution (e.g., for surrogate-based optimization) so that multiple executions are repeatable but not identical.

- int numCVTTrials

  specifies the number of sample points taken at internal CVT iteration

- int trialType

  Trial type in CVT. Specifies where the points are placed for consideration relative to the centroids. Choices are grid (2), halton (1), uniform (0), or random (-1). Default is random.

10.39.1 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.

10.39.2 Constructor & Destructor Documentation

10.39.2.1 FSUDesignCompExp (Model & *model*)

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

10.39.3 Member Function Documentation

10.39.3.1 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

10.40 FunctionCompare Class Template Reference

Public Member Functions

- FunctionCompare (bool(∗func)(const T &, void ∗), void ∗v)

  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.

10.40.1 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.

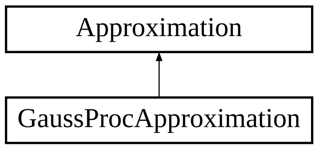
The documentation for this class was generated from the following file:

- DakotaList.H

10.41 GaussProcApproximation Class Reference

Derived approximation class for Gaussian Process implementation.

Inheritance diagram for 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

    return the minimum number of samples required to build the 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

Private Member Functions

- void GPmodel_build ()

  Function to compute hyperparameters governing the GP.

- Real GPmodel_apply (const RealVector &new_x)

  Function returns a response value using the GP surface.

- void predict ()

  Calculates the predicted new response value for x in normalized space.

- void normalize ()

  Normalizes the initial inputs upon which the GP surface is based.

- void unnormalize ()

  Takes the value from the GP normalized prediction and unnormalizes it.

- Real calc_nll ()

  calculates the negative log likelihood function (based on covariance matrix)

- void covmatrix ()

  calculates the covariance matrix for a given set of input points

- void covvector ()

  calculates the covariance vector between a new point x and the set of inputs upon which the GP is based

- void optimize_nll ()

  function which sets up and performs the optimization of the negative log likelihood to determine the optimal values of the covariance oaraneters

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)

  static function used by OPT++ as the objective function to optimize the hyperparameters in the covariance of the GP by minimizing the negative log likelihood

- static void constraint_eval (int mode, int n, const NEWMAT::ColumnVector &X, NEWMAT::ColumnVector &g, NEWMAT::Matrix &gradC, int &result_mode)

  static function used by OPT++ as the constraint function in the optimization of the negative log likelihood. Currently this function is empty: it is an unconstrained optimization.

Private Attributes

- Epetra_SerialDenseMatrix x_matrix

  A 2-D array (num sample sites = rows, num vars = columns) used to create the Gaussian process.

- Epetra_SerialDenseMatrix f_of_x_array

  An array of response values; one response value per sample site.

- Epetra_SerialDenseMatrix initX

  Initial set of sample values of X upon which the GP is based.

- Epetra_SerialDenseMatrix newX

  New value of x at which one wants a point prediction. This is currently a single point, but it may be a vector of X values, where each X can be multi-dimensional.

- Epetra_SerialDenseMatrix outY

  output Y corresponding to initX

- Epetra_SerialDenseMatrix covmatrixX

  The covariance matrix where each element (i,j) is the covariance between points Xi and Xj in the initial set of samples.

- Epetra_SerialDenseMatrix covvectorX

  The covariance vector where each element (j,0) is the covariance between a new point X and point Xj from the initial set of samples.

- Epetra_SerialDenseMatrix newY

  The Gaussian process prediction for point newX.

- Epetra_SerialDenseMatrix tempholder

  A temporary placeholder matrix to allow for Epetra matrix multiplication.

- Epetra_SerialDenseMatrix cov_mult

  Another temporary placeholder matrix to allow for Epetra matrix multiplication.

- Epetra_SerialDenseVector mean_column

  The mean of the input columns of initX.

- Epetra_SerialDenseVector sqterms

  The standard deviation of the input columns of initX.

- Real meanY

  The mean of the output Y.

- Real stdY

  The standard deviation of the output Y.

- size_t num_obs

  The number of observations on which the GP surface is built.

- size_t num_vars

  The number of variables in each X variable (number of dimensions of the problem).

- size_t num_new

  The number of new X values for which one wants a prediction. In this implementation, num_new = 1.

- Real sige

  The GP error term.

- RealVector theta

  Theta is the vector of covariance parameters for the GP. We determine the values of theta by optimization Currently, the covariance function is theta[0]∗exp(-0.5∗sume)+delta∗pow(sige,2). sume is the sum squared of weighted distances; it involves a sum of theta[1](Xi(1)-Xj(1))^2 + theta[2](Xi(2)-Xj(2))^2 + ... where Xi(1) is the first dimension value of multi-dimensional variable Xi. delta∗pow(sige,2) is a jitter term used to improve matrix computations. delta is zero for the covariance between different points and 1 for the covariance between the same point. sige is the underlying process error.

- Real nll

  The negative log likelihood value for a particular matrix.

- Real approxValue

  A placeholder to return the Gaussian process prediction.

Static Private Attributes

- static GaussProcApproximation ∗ GPinstance

  pointer to the active object instance used within the static evaluator

10.41.1 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.

10.41.2 Member Function Documentation

10.41.2.1 Real GPmodel_apply (const RealVector & *new_x*) `[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.

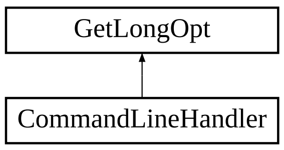
The documentation for this class was generated from the following files:

- GaussProcApproximation.H
- GaussProcApproximation.C

10.42 GetLongOpt Class Reference

GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

Inheritance diagram for GetLongOpt::



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

  extract the base name from a string as delimited by '/'

- int setcell (Cell ∗c, char ∗valtoken, char ∗nexttoken, const char ∗p)

  internal convenience function for setting Cell::value

Private Attributes

- Cell ∗ table

  option table

- const char ∗ ustring

  usage message

- char ∗ pname

  program basename

- char optmarker

  option marker

- int enroll_done

  finished enrolling

- Cell ∗ last

  last entry in option table

10.42.1 Detailed Description

GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

GetLongOpt manages the definition and parsing of "long options." Command line options can be abbreviated as long as there is no ambiguity. If an option requires a value, the value should be separated from the option either by whitespace or an "=".

10.42.2 Constructor & Destructor Documentation

10.42.2.1 GetLongOpt (const char *optmark* = ' - ')

Constructor.

Constructor for GetLongOpt takes an optional argument: the option marker. If unspecified, this defaults to '-', the standard (?) Unix option marker.

10.42.3 Member Function Documentation

10.42.3.1 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.

10.42.3.2 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.

10.42.3.3 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.

10.42.3.4 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.

10.42.3.5 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

10.43 Graphics Class Reference

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

Public Member Functions

- Graphics ()

  constructor
- ∼Graphics ()

  destructor
- void create_plots_2d (const Variables &vars, const Response &response)

  creates the 2d graphics window and initializes the plots
- void create_tabular_datastream (const Variables &vars, const Response &response, const String &tabular_-data_file)

  opens the tabular data file stream and prints the headings
- void add_datapoint (const Variables &vars, const Response &response)

  adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model evaluation
- void add_datapoint (int i, double x, double y)

  adds data to a single window in the 2d graphics
- void new_dataset (int i)

  creates a separate line graphic for subsequent data points for a single window in the 2d graphics
- void 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

10.43.1 Detailed Description

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

There is only one Graphics object (dakotaGraphics) and it is global (for convenient access from strategies, models, and approximations).

10.43.2 Member Function Documentation

10.43.2.1 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).

10.43.2.2 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.

10.43.2.3 void add_datapoint (const Variables & *vars*, const Response & *response*)

adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model evaluation

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.

10.43.2.4 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.

10.43.2.5 void new_dataset (int *i*)

creates a separate line graphic for subsequent data points for a single window in the 2d graphics

Used for displaying multiple data sets within the same plot.

10.43.2.6 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.

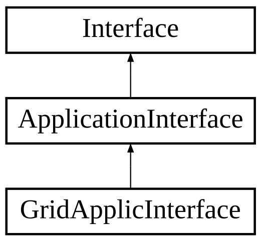
The documentation for this class was generated from the following files:

- DakotaGraphics.H
- DakotaGraphics.C

10.44 GridApplicInterface Class Reference

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.

Inheritance diagram for 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)

  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.

- void derived_map_asynch (const ParamResponsePair &pair)

  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.

- void derived_synch (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- void derived_synch_nowait (PRPList &prp_list)

  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- int 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

  Set of function evaluation id's for active asynchronous system call evaluations.

- IntShortMap failCountMap

  map linking function evaluation id's to number of response read failures

- start_grid_computing_t start_grid_computing

  handle to dynamically linked start_grid_computing function

- perform_analysis_t perform_analysis

  handle to dynamically linked perform_analysis grid function

- get_jobs_completed_t get_jobs_completed

  handle to dynamically linked get_jobs_completed grid function

- stop_grid_computing_t stop_grid_computing

  handle to dynamically linked stop_grid_computing function

10.44.1 Detailed Description

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.

This class is currently a modified copy of SysCallApplicInterface adapted for use with an external grid devices library which was dynamically linked using dlopen() services.

10.44.2 Member Function Documentation

10.44.2.1 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.

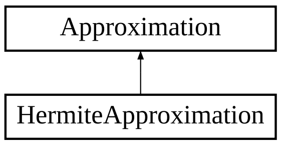
The documentation for this class was generated from the following files:

- GridApplicInterface.H
- GridApplicInterface.C

10.45 HermiteApproximation Class Reference

Derived approximation class for Hermite polynomials (global approximation).

Inheritance diagram for HermiteApproximation::



Public Member Functions

- HermiteApproximation ()

    default constructor

- HermiteApproximation (ProblemDescDB &problem_db, const size_t &num_acv)

    standard constructor

- ∼HermiteApproximation ()

    destructor

Protected Member Functions

- int num_coefficients () const

    return the minimum number of samples required to build the 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 ()

    return the coefficient array computed by find_coefficients()

- void find_coefficients ()

    find the Polynomial Chaos coefficients for the response surface

- const Real & get_value (const RealVector &x)

    retrieve the function value for a given parameter set x

Private Member Functions

- void get_num_chaos ()

  calculate number of Chaos according to the highest order of Chaos

- RealVector get_chaos (const RealVector &x, int order)

  calculate the Polynomial Chaos from variables

Private Attributes

- RealVector chaosCoeffs

  numChaos entries

- RealVectorArray chaosSamples

  numChaos∗*num_pts entries*

- int numChaos

  Number of terms in Polynomial Chaos Expansion.

- int highestOrder

  Highest order of Hermite Polynomials in Expansion.

10.45.1 Detailed Description

Derived approximation class for Hermite polynomials (global approximation).

The HermiteApproximation class provides a global approximation based on Hermite polynomials. It is used primarily for polynomial chaos expansions (for stochastic finite element approaches to uncertainty quantification).

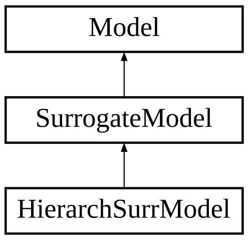
The documentation for this class was generated from the following files:

- HermiteApproximation.H
- HermiteApproximation.C

10.46 HierarchSurrModel Class Reference

Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

Inheritance diagram for 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)

  set surrogateBypass flag and pass request on to highFidelityModel for any lower-level surrogates.

- void build_approximation ()

  use highFidelityModel to compute the truth values needed for correction of lowFidelityModel results

- void component_parallel_mode (int mode)

  update component parallel mode for supporting parallelism in lowFidelityModel and highFidelityModel

- void derived_init_communicators (const int &max_iterator_concurrency)

  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)

  set active parallel configuration within lowFidelityModel and highFidelityModel

- void reset_communicators ()

  reset communicator partition data for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelityModel)

- void derived_free_communicators (const int &max_iterator_concurrency)

  deallocate communicator partitions for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelityModel)

- void serve ()

  Service lowFidelityModel and highFidelityModel job requests received from the master. Completes when a termination message is received from stop_servers().

- void stop_servers ()

  Executed by the master to terminate lowFidelityModel and highFidelityModel server operations when iteration on the HierarchSurrModel is complete.

- int evaluation_id () const

  Return the current evaluation id for the HierarchSurrModel.

- void set_evaluation_reference ()

  set the evaluation counter reference points for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelityModel)

- void print_evaluation_summary (ostream &s, bool minimal_header=false, bool relative_count=true) const

  print the evaluation summary for the HierarchSurrModel (request forwarded to lowFidelityModel and highFidelity-Model)

Private Member Functions

- void update_model (Model &model)

  update the incoming model (lowFidelityModel or highFidelityModel) with current variable values/bounds/labels

Private Attributes

- int hierModelEvals

  number of calls to derived_compute_response()/ derived_asynch_compute_response()

- Model lowFidelityModel

  provides approximate low fidelity function evaluations. Model is of arbitrary type and supports recursions (e.g., lowFidelityModel can be a data fit surrogate on a low fidelity model).

- Model highFidelityModel

  provides truth evaluations for computing corrections to the low fidelity results. Model is of arbitrary type and supports recursions.

- Response highFidResponse

  the high fidelity response is computed in build_approximation() and needs class scope for use in automatic surrogate construction in derived compute_response functions.

10.46.1 Detailed Description

Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

The HierarchSurrModel class manages hierarchical models of varying fidelity. In particular, it uses a low fidelity model as a surrogate for a high fidelity model. The class contains a lowFidelityModel which performs the approximate low fidelity function evaluations and a highFidelityModel which provides truth evaluations for computing corrections to the low fidelity results.

10.46.2 Member Function Documentation

10.46.2.1 void derived_compute_response (const ActiveSet & *set*) [protected, virtual]

portion of compute_response() specific to HierarchSurrModel

Evaluate the approximate response using lowFidelityModel, compute the high fidelity response if needed with build_approximation(), and, if correction is active, correct the low fidelity results.

Reimplemented from Model.

10.46.2.2 void derived_asynch_compute_response (const ActiveSet & *set*) [protected, virtual]

portion of asynch_compute_response() specific to HierarchSurrModel

Evaluate the approximate response using an asynchronous lowFidelityModel evaluation and 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.

10.46.2.3 const ResponseArray & derived_synchronize () [protected, virtual]

portion of synchronize() specific to HierarchSurrModel

Perform a blocking retrieval of all asynchronous evaluations from lowFidelityModel and, if automatic correction is on, apply correction to each response in the array.

Reimplemented from Model.

10.46.2.4 const IntResponseMap & derived_synchronize_nowait () [protected, virtual]

portion of synchronize_nowait() specific to HierarchSurrModel

Perform a nonblocking retrieval of currently available asynchronous evaluations from lowFidelityModel and, if automatic correction is on, apply correction to each response in the list.

Reimplemented from Model.

10.46.2.5 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.

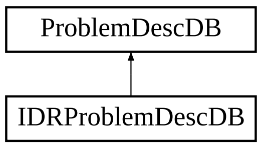
The documentation for this class was generated from the following files:

- HierarchSurrModel.H
- HierarchSurrModel.C

10.47 IDRProblemDescDB Class Reference

The derived input file database utilizing the IDR parser.

Inheritance diagram for IDRProblemDescDB::



Public Member Functions

- IDRProblemDescDB (ParallelLibrary &parallel_lib)
  constructor
- ~IDRProblemDescDB ()
  destructor
- void derived_manage_inputs (const char ∗dakota_input_file)
  parses the input file and populates the problem description database using IDR.

Static Public Member Functions

- static void strategy_kwhandler (const struct FunctionData ∗parsed_data)
  strategy keyword handler called by IDR when a complete strategy specification is parsed
- static void method_kwhandler (const struct FunctionData ∗parsed_data)
  method keyword handler called by IDR when a complete method specification is parsed
- static void model_kwhandler (const struct FunctionData ∗parsed_data)
  model keyword handler called by IDR when a complete model specification is parsed
- static void variables_kwhandler (const struct FunctionData ∗parsed_data)
  variables keyword handler called by IDR when a complete variables specification is parsed
- static void interface_kwhandler (const struct FunctionData ∗parsed_data)
  interface keyword handler called by IDR when a complete interface specification is parsed
- static void responses_kwhandler (const struct FunctionData ∗parsed_data)
  responses keyword handler called by IDR when a complete 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)

    Function used by the keyword handlers to return the number of parsed 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

    pointer to the active object instance used within the static kwhandler 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()

10.47.1 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

10.47.2 Member Function Documentation

10.47.2.1 void derived_manage_inputs (const char * *dakota_input_file*) `[virtual]`

parses the input file and populates the problem description 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.

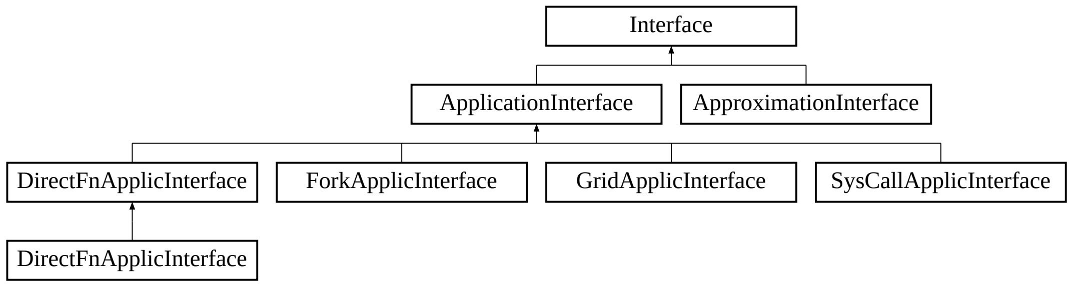
The documentation for this class was generated from the following files:

- IDRProblemDescDB.H
- IDRProblemDescDB.C

10.48 Interface Class Reference

Base class for the interface class hierarchy.

Inheritance diagram for Interface::



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)

    the function evaluator: provides a "mapping" from the variables to the responses.

- virtual const ResponseArray & synch ()

    recovers data from a series of asynchronous evaluations (blocking)

- virtual const IntResponseMap & 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)

  allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- virtual void reset_communicators (const IntArray &message_lengths)

  reset the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

- virtual void free_communicators ()

  deallocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

- virtual void init_serial ()

  reset certain defaults for serial interface objects.

- virtual int asynch_local_evaluation_concurrency () const

  return the user-specified concurrency for asynch local evaluations

- virtual String interface_synchronization () const

  return the user-specified interface synchronization

- virtual int minimum_samples (bool constraint_flag) const

  returns the minimum number of samples required to build a particular ApproximationInterface (used by DataFit-SurrModels).

- virtual void update_approximation (const RealVectorArray &all_variables, const ResponseArray &all_-responses)

  passes multiple points to an approximation for building a surrogate

- virtual void update_approximation (const RealVector &c_variables, const Response &response)

  passes a single point to an approximation for building a surrogate

- virtual void build_approximation (const RealVector &lower_bnds, const RealVector &upper_bnds)

  builds the surrogate

- virtual void append_approximation (const RealVector &c_variables, const Response &response)

  updates an existing global approximation with new data

- virtual void clear ()

  clears all data from an approximation interface

- virtual bool anchor () const

  queries the presence of an anchorPoint within an approximation interface

- virtual const RealVectorArray & approximation_coefficients ()

  retrieve the approximation coefficients from each Approximation 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

  returns a flag signaling the use of a dedicated master processor at the iterator-evaluation scheduling level

- bool is_null () const

  function to check interfaceRep (does this envelope contain a letter?)

Protected Member Functions

- Interface (BaseConstructor, const ProblemDescDB &problem_db)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- void asv_mapping (const ActiveSet &total_set, ActiveSet &algebraic_set, ActiveSet &core_set, const Variables &vars, const Response &response)

define the evaluation requirements for algebraic_mappings() (algebraic_set) and the core Application/Approximation mapping (core_set) from the total Interface evaluation requirements (total_set). Also

- void algebraic_mappings (const Variables &vars, const ActiveSet &algebraic_set, Response &algebraic_-response)

  evaluate the algebraic_response using the AMPL solver library and the data extracted from the algebraic_mappings file

- void response_mapping (const Response &algebraic_response, const Response &core_response, Response &total_response)

  combine the response from algebraic_mappings() with the response from derived_map() to create the total response

Protected Attributes

- String interfaceType

  the interface type: system, fork, direct, grid, or approximation

- String idInterface

  the interface specification indentifier string from the DAKOTA input file (used in print_eval_summary())

- bool algebraicMappings

  flag for the presence of algebraic_mappings that define the subset of an Interface's parameter to response mapping that is explicit and algebraic.

- bool coreMappings

  flag for the presence of non-algebraic mappings that define the core of an Interface's parameter to response mapping (using analysis_drivers for ApplicationInterface or functionSurfaces for ApproximationInterface).

- int 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

  The complete array of responses returned after a blocking schedule of asynchronous evaluations.

- IntResponseMap rawResponseMap

  The partial map of responses returned after a nonblocking schedule of asynchronous evaluations.

- StringArray responseTags

  response function identifier tags from the DAKOTA input file (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.

Private Attributes

- StringArray algebraicVarTags

  set of variable tags from AMPL stub.col

- SizetArray algebraicACVIndices

  set of indices mapping AMPL algebraic variables to DAKOTA all continuous variables

- SizetArray algebraicDerivIndices

  set of indices mapping AMPL algebraic variables to DAKOTA derivative variables

- StringArray algebraicFnTags

  set of function tags from AMPL stub.row

- SizetArray algebraicFnIndices

  set of indices mapping AMPL algebraic objective functions to DAKOTA response functions

- Interface ∗ interfaceRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing interfaceRep

10.48.1 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.

10.48.2 Constructor & Destructor Documentation

10.48.2.1 Interface ()

default constructor

used in Model envelope class instantiations

10.48.2.2 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.

10.48.2.3 Interface (const Interface & *interface*)

copy constructor

Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.

10.48.2.4 ~Interface () `[virtual]`

destructor

Destructor decrements referenceCount and only deletes interfaceRep if referenceCount is zero.

10.48.2.5 Interface (BaseConstructor, const ProblemDescDB & *problem_db*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all inherited interfaces. get_interface() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_interface() again). Since this is the letter and the letter IS the representation, interfaceRep is set to NULL (an uninitialized pointer causes problems in ~Interface).

10.48.3 Member Function Documentation

10.48.3.1 Interface operator= (const Interface & *interface*)

assignment operator

Assignment operator decrements referenceCount for old interfaceRep, assigns new interfaceRep, and increments referenceCount for new interfaceRep.

10.48.3.2 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).

10.48.3.3 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.

10.48.4 Member Data Documentation

10.48.4.1 ResponseArray rawResponseArray `[protected]`

The complete array of responses returned after a blocking schedule of 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.

10.48.4.2 IntResponseMap rawResponseMap `[protected]`

The partial map of responses returned after a nonblocking schedule of 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().

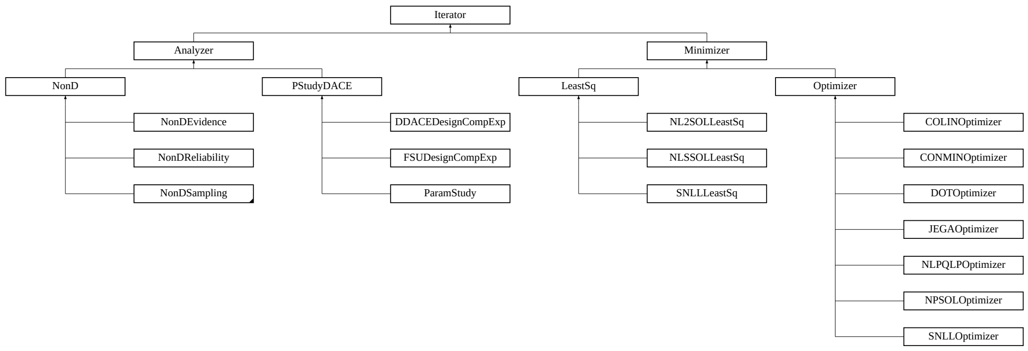
The documentation for this class was generated from the following files:

- DakotaInterface.H
- DakotaInterface.C

10.49 Iterator Class Reference

Base class for the iterator class hierarchy.

Inheritance diagram for Iterator::



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 & variable_results () const

  return the final iterator solution (variables)

- virtual const Response & response_results () const

  return the final iterator solution (response)

- 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)

  set the relative weightings for multiple objective functions. 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 RealVectorArray & all_c_variables () const

  return the complete set of evaluated continuous variables

- virtual const ResponseArray & all_responses () const

  return the complete set of computed responses

- virtual const RealVectorArray & all_fn_responses () const

  return the complete set of computed function responses

- void pre_run ()

  utility function to 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 perform common operations following run() verbosely

- void post_run ()

  utility function to perform common operations following run() quietly

- void assign_rep (Iterator ∗iterator_rep, bool ref_count_incr=true)

  replaces existing letter with a new one

- void user_defined_model (const Model &the_model)

  set the model

- Model & user_defined_model () const

  return the model

- 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)

  set the default active set vector (for use with iterators that employ evaluate_parameter_sets())

- const ActiveSet & active_set () const

  return the default active set vector (used by iterators that employ evaluate_parameter_sets())

- void sub_iterator_flag (bool si_flag)

  set subIteratorFlag

- void variable_mappings (const SizetArray &c_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?)

Protected Member Functions

- Iterator (BaseConstructor, Model &model)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- Iterator (NoDBBaseConstructor, Model &model)

  base class for iterator classes constructed on the fly (no DB queries)

- 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

Protected Attributes

- Model & userDefinedModel

  class member reference for the model passed into the constructor

- 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

- 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

  flag indicating if this Iterator is a sub-iterator (NestedModel::subIterator or DataFitSurrModel::daceIterator)

- SizetArray primaryCVarMapIndices

  "primary" continuous variable mappings flowed down from higher level iteration

- SizetArray primaryDVarMapIndices

  "primary" discrete variable mappings flowed down from higher 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 intervalType

  type of numerical gradient interval: central or forward

- String methodSource

  source of numerical gradient routine: dakota or vendor

- String hessianType

  type of Hessian data: analytic, numerical, quasi, mixed, or none

- Real fdGradStepSize

  relative finite difference step size for numerical gradients

- Real fdHessByGradStepSize

  relative finite difference step size for numerical Hessians estimated using first-order differences of gradients

- Real fdHessByFnStepSize

  relative finite difference step size for numerical Hessians estimated using second-order differences of function values

- bool silentOutput

  flag for really quiet (silent) algorithm output

- bool quietOutput

  flag for quiet algorithm output

- bool verboseOutput

  flag for verbose algorithm output

- bool debugOutput

  flag for really verbose (debug) algorithm output

- bool asynchFlag

  copy of the model's asynchronous evaluation flag

Private Member Functions

- Iterator ∗ get_iterator (Model &model)

  Used by the envelope to instantiate the correct letter class.

Private Attributes

- String idMethod

  method identifier string from the input file

- Iterator ∗ iteratorRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing iteratorRep

10.49.1 Detailed Description

Base class for the iterator class hierarchy.

The Iterator class is the base class for one of the primary class hierarchies in DAKOTA. The iterator hierarchy contains all of the iterative algorithms which use repeated execution of simulations as function evaluations. For memory efficiency and enhanced polymorphism, the iterator hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Iterator) serves as the envelope and one of the derived classes (selected in Iterator::get_iterator()) serves as the letter.

10.49.2 Constructor & Destructor Documentation

10.49.2.1 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.

10.49.2.2 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.

10.49.2.3 Iterator (const Iterator & *iterator*)

copy constructor

Copy constructor manages sharing of iteratorRep and incrementing of referenceCount.

10.49.2.4 ∼Iterator () `[virtual]`

destructor

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

10.49.2.5 Iterator (BaseConstructor, Model & *model*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor builds the base class data for all inherited iterators. get_iterator() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_iterator() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ∼Iterator).

10.49.2.6 Iterator (NoDBBaseConstructor, Model & *model*) `[protected]`

base class for iterator classes constructed on the fly (no DB queries)

This constructor also builds base class data for inherited iterators. However, 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.

10.49.3 Member Function Documentation

10.49.3.1 Iterator operator= (const Iterator & *iterator*)

assignment operator

Assignment operator decrements referenceCount for old iteratorRep, assigns new iteratorRep, and increments referenceCount for new iteratorRep.

10.49.3.2 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.

10.49.3.3 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, NonDLHSSampling, NonDPCESampling, and NonDReliability.

10.49.3.4 void pre_run ()

utility function to perform common operations prior to run()

Iterator supports a construct/pre-run/run/post-run/destruct progression. This function is the pre-run function. This function is not virtual: derived portions are defined in derived_pre_run().

10.49.3.5 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.

10.49.3.6 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.

10.49.3.7 void post_run (ostream & *s*)

utility function to perform common operations following run() verbosely

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().

10.49.3.8 void post_run ()

utility function to perform common operations following run() quietly

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().

10.49.3.9 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).

10.49.3.10 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, SNLLLeastSq, and SNLLOptimizer.

10.49.3.11 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 CONMINOptimizer, DOTOptimizer, NLPQLPOptimizer, SNLLLeastSq, and SNLLOptimizer.

10.49.3.12 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.

10.49.4 Member Data Documentation

10.49.4.1 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.

10.49.4.2 Real fdHessByGradStepSize `[protected]`

relative finite difference step size for numerical Hessians estimated using first-order differences of gradients

A scalar value (instead of the vector fd_hessian_step_size spec) is used within the iterator hierarchy since this attribute is only used to publish a step size to vendor numerical Hessian algorithms.

10.49.4.3 Real fdHessByFnStepSize `[protected]`

relative finite difference step size for numerical Hessians estimated using second-order differences of function values

A scalar value (instead of the vector fd_hessian_step_size spec) is used within the iterator hierarchy since this attribute is only used to publish a step size to vendor numerical Hessian algorithms.

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

- DakotaIterator.H
- DakotaIterator.C

10.50 JEGAEvaluator Class Reference

This evaluator uses Sandia National Laboratories Dakota software.

Public Member Functions

- virtual bool Evaluate (JEGA::Utilities::DesignGroup &group)

  Does evaluation of each design in "group'.

- virtual bool Evaluate (JEGA::Utilities::Design &des)

  This method cannot be used!!

- virtual std::string GetName () const

  Returns the proper name of this operator.

- virtual std::string GetDescription () const

  Returns a full description of what this operator does and how.

- virtual GeneticAlgorithmOperator ∗ Clone (JEGA::Algorithms::GeneticAlgorithm &algorithm) const

  Creates and returns a pointer to an exact duplicate of this operator.

- JEGAEvaluator (JEGA::Algorithms::GeneticAlgorithm &algorithm, Model &model, JEGAOptimizer &theOptimizer)

  Constructs a JEGAEvaluator for use by algorithm.

- JEGAEvaluator (const JEGAEvaluator &copy)

  Copy constructs a JEGAEvaluator.

- JEGAEvaluator (const JEGAEvaluator &copy, JEGA::Algorithms::GeneticAlgorithm &algorithm, Model &model, JEGAOptimizer &theOptimizer)

  Copy constructs a JEGAEvaluator for use by algorithm.

Static Public Member Functions

- static const std::string & Name ()

  Returns the proper name of this operator.

- static const std::string & Description ()

  Returns a full description of what this operator does and how.

Protected Member Functions

- RealVector GetContinuumVariableValues (const JEGA::Utilities::Design &from) const

  Returns the continuous Design variable values held in Design from.

- IntVector GetDiscreteVariableValues (const JEGA::Utilities::Design &from) const

  Returns the discrete Design variable values held in Design from.

- void GetContinuumVariableValues (const JEGA::Utilities::Design &from, RealVector &into) const

  Places the continuous Design variable values from Design from *into RealVector* into.

- void GetDiscreteVariableValues (const JEGA::Utilities::Design &from, IntVector &into) const

  Places the discrete Design variable values from Design from *into IntVector* into.

- void SeparateVariables (const JEGA::Utilities::Design &from, IntVector &intoDisc, RealVector &intoCont) const

  This method fills intoDisc *and* intoCont *appropriately using the values of* from.

- void RecordResponses (const RealVector &from, JEGA::Utilities::Design &into) const

  Records the computed objective and constraint function values into into.

- std::size_t GetNumberNonLinearConstraints () const

  Returns the number of non-linear constraints for the problem.

- std::size_t GetNumberLinearConstraints () const

  Returns the number of linear constraints for the problem.

Private Member Functions

- JEGAEvaluator (JEGA::Algorithms::GeneticAlgorithm &algorithm)

  This constructor has no implementation and cannot be used.

Private Attributes

- Model & _model

  The Model known by this evaluator.

- JEGAOptimizer & _theOptimizer

  The JEGAOptimizer that created this evaluator.

10.50.1 Detailed Description

This evaluator uses Sandia National Laboratories Dakota software.

Evaluations are carried out using a Model which is known by reference to this class. This provides the advantage of execution on massively parallel computing architectures.

10.50.2 Constructor & Destructor Documentation

10.50.2.1 JEGAEvaluator (JEGA::Algorithms::GeneticAlgorithm & *algorithm*, Model & *model*, JEGAOptimizer & *theOptimizer*)

Constructs a JEGAEvaluator for use by *algorithm*.

The optimizer is needed for purposes of variable scaling.

Parameters:

algorithm The GA for which the new evaluator is to be used.

model The model through which evaluations will be done.

theOptimizer The optimizer that created and is using this evaluator.

10.50.2.2 JEGAEvaluator (const JEGAEvaluator & *copy*)

Copy constructs a JEGAEvaluator.

Parameters:

copy The evaluator from which properties are to be duplicated into this.

10.50.2.3 JEGAEvaluator (const JEGAEvaluator & *copy*, JEGA::Algorithms::GeneticAlgorithm & *algorithm*, Model & *model*, JEGAOptimizer & *theOptimizer*)

Copy constructs a JEGAEvaluator for use by *algorithm*.

The optimizer is needed for purposes of variable scaling.

Parameters:

copy The existing JEGAEvaluator from which to retrieve properties.

algorithm The GA for which the new evaluator is to be used.

model The model through which evaluations will be done.

theOptimizer The optimizer that created and is using this evaluator.

10.50.2.4 JEGAEvaluator (JEGA::Algorithms::GeneticAlgorithm & *algorithm*) `[private]`

This constructor has no implementation and cannot be used.

This constructor can never be used. It is provided so that this operator can still be registered in an operator registry even though it can never be instantiated from there.

Parameters:

algorithm The GA for which the new evaluator is to be used.

10.50.3 Member Function Documentation

10.50.3.1 const string & Name () `[static]`

Returns the proper name of this operator.

Returns:

The string "JEGA Evaluator".

10.50.3.2 const string & Description () `[static]`

Returns a full description of what this operator does and how.

The returned text is:

```
This evaluator uses Sandia's DAKOTA optimization
software to evaluate the passed in Designs.  This
makes it possible to take advantage of the fact that
DAKOTA is designed to run on massively parallel machines.
```

.

Returns:

A description of the operation of this operator.

10.50.3.3 RealVector GetContinuumVariableValues (const JEGA::Utilities::Design & *from*) const `[protected]`

Returns the continuous Design variable values held in Design *from*.

It returns them as a RealVector for use in the Dakota interface. The values in the returned vector will be the actual values intended for use in the evaluation functions.

Parameters:

from The Design class object from which to extract the continuous design variable values.

Returns:

A vector of the continuous design variable values associated with *from*.

10.50.3.4 IntVector GetDiscreteVariableValues (const JEGA::Utilities::Design & *from*) const `[protected]`

Returns the discrete Design variable values held in Design *from*.

It returns them as a IntVector for use in the Dakota interface. The values in the returned vector will be the values for the design variables as far as JEGA knows. However, in actuality, the values are the representations due to the way that Dakota manages discrete variables.

Parameters:

from The Design class object from which to extract the discrete design variable values.

Returns:

A vector of the discrete design variable values associated with *from*.

10.50.3.5 void GetContinuumVariableValues (const JEGA::Utilities::Design & *from*, RealVector & *into*) const `[protected]`

Places the continuous Design variable values from Design *from* into RealVector *into*.

The values in the returned vector will be the actual values intended for use in the evaluation functions.

Parameters:

from The Design class object from which to extract the continuous design variable values.

into The vector into which to place the extracted values.

10.50.3.6 void GetDiscreteVariableValues (const JEGA::Utilities::Design & *from*, IntVector & *into*) const `[protected]`

Places the discrete Design variable values from Design *from* into IntVector *into*.

The values placed in the vector will be the values for the design variables as far as JEGA knows. However, in actuality, the values are the representations due to the way that Dakota manages discrete variables.

Parameters:

from The Design class object from which to extract the discrete design variable values.

into The vector into which to place the extracted values.

10.50.3.7 void SeparateVariables (const JEGA::Utilities::Design & *from*, IntVector & *intoDisc*, RealVector & *intoCont*) const `[protected]`

This method fills *intoDisc* and *intoCont* appropriately using the values of *from*.

The discrete design variable values are placed in *intoDisc* and the continuum are placed into *intoCont*.

It is more efficient to use this method than to use GetDiscreateVariableValues and GetContinuumVariableValues separately if you want both.

Parameters:

from The Design class object from which to extract the discrete design variable values.

intoDisc The vector into which to place the extracted discrete values.

intoCont The vector into which to place the extracted continuous values.

10.50.3.8 void RecordResponses (const RealVector & *from*, JEGA::Utilities::Design & *into*) const `[protected]`

Records the computed objective and constraint function values into *into*.

This method takes the response values stored in *from* and properly transfers them into the *into* design.

The response vector *from* is expected to contain values for each objective function followed by values for each non-linear constraint in the order in which the info objects were loaded into the target by the optimizer class.

Parameters:

from The vector of responses to install into *into*.

into The Design to which the responses belong and into which they must be written.

10.50.3.9 size_t GetNumberNonLinearConstraints () const `[protected]`

Returns the number of non-linear constraints for the problem.

This is computed by adding the number of non-linear equality constraints to the number of non-linear inequality constraints. These values are obtained from the model.

Returns:

The total number of non-linear constraints.

10.50.3.10 size_t GetNumberLinearConstraints () const `[protected]`

Returns the number of linear constraints for the problem.

This is computed by adding the number of linear equality constraints to the number of linear inequality constraints. These values are obtained from the model.

Returns:

The total number of linear constraints.

10.50.3.11 virtual bool Evaluate (JEGA::Utilities::DesignGroup & *group*) `[virtual]`

Does evaluation of each design in "group'.

This method uses the Model known by this class to get Designs evaluated. It properly formats the Design class information in a way that Dakota will understand and then interprets the Dakota results and puts them back into the Design class object. It respects the asynchronous flag in the Model so evaluations may occur synchronously or asynchronously.

Prior to evaluating a Design, this class checks to see if it is marked as already evaluated. If it is, then the evaluation of that Design is not carried out. This is not strictly necessary because Dakota keeps track of evaluated designs and does not re-evaluate. An exception is the case of a population read in from a file complete with responses where Dakota is unaware of the evaluations.

Parameters:

group The group of Design class objects to be evaluated.

Returns:

true if all evaluations completed and false otherwise.

10.50.3.12 virtual bool Evaluate (JEGA::Utilities::Design & *des*) `[virtual]`

This method cannot be used!!

This method does nothing and cannot be called. This is because in the case of asynchronous evaluation, this method would be unable to conform. It would require that each evaluation be done in a synchronous fashion.

Parameters:

des A Design that would be evaluated if this method worked.

Returns:

Would return true if the Design were evaluated and false otherwise.

10.50.3.13 string GetName () const `[virtual]`

Returns the proper name of this operator.

Returns:

See Name().

10.50.3.14 string GetDescription () const `[virtual]`

Returns a full description of what this operator does and how.

Returns:
See Description().

10.50.3.15 virtual GeneticAlgorithmOperator* Clone (JEGA::Algorithms::GeneticAlgorithm & *algorithm*) const `[virtual]`

Creates and returns a pointer to an exact duplicate of this operator.

Parameters:
algorithm The GA for which the clone is being created.

Returns:
A clone of this operator.

10.50.4 Member Data Documentation

10.50.4.1 Model& _model `[private]`

The Model known by this evaluator.

It is through this model that evaluations will take place.

10.50.4.2 JEGAOptimizer& _theOptimizer `[private]`

The JEGAOptimizer that created this evaluator.

This instance is used to access certain needed functions from the optimizer class such as those methods that do variable scaling.

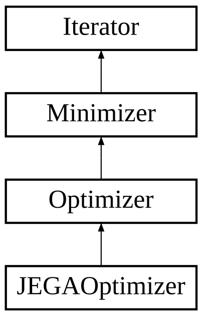
The documentation for this class was generated from the following files:

- JEGAEvaluator.H
- JEGAEvaluator.C

10.51 JEGAOptimizer Class Reference

Version of Optimizer for instantiation of John Eddy's Genetic Algorithms.

Inheritance diagram for JEGAOptimizer::



Public Member Functions

- virtual void find_optimum ()

  Performs the iterations to determine the optimal set of solutions.

- JEGAOptimizer (Model &model)

  Constructs a JEGAOptimizer class object.

- ~JEGAOptimizer ()

  Destructs a JEGAOptimizer.

Protected Member Functions

- void ReCreateTheAlgorithmConfig ()

  Destroys the existing algorithm configuration and creates a new one.

- void ReCreateTheProblemConfig ()

  Destroys the existing problem configuration and creates a new one.

- void LoadTheAlgorithmConfig ()

  Completely initializes or re-initializes the algorithm configuration.

- void LoadTheProblemConfig ()

  Completely initializes or re-initializes the problem configuration.

- void LoadTheDesignVariables ()

  Adds DesignVariableInfo objects into the problem configuration object.

- void LoadTheObjectiveFunctions ()

  Adds ObjectiveFunctionInfo objects into the problem configuration object.

- void LoadTheConstraints ()

  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.

Private Attributes

- EvalCreator ∗ _theEvalCreator

  A pointer to an EvaluatorCreator used to create the evaluator used by JEGA in Dakota (a JEGAEvaluator).

- JEGA::FrontEnd::ProblemConfig ∗ _theProbConfig

  A pointer to the problem configuration loaded by this optimizer at each call to find_optimum and passed to the JEGA Driver.

- JEGA::FrontEnd::AlgorithmConfig ∗ _theAlgConfig

  A pointer to the algorithm configuration loaded by this optimizer at each call to find_optimum and passed to the JEGA Driver.

- JEGAProbDescDB ∗ _theProbDB

  A pointer to the JEGAProbDescDB that wraps the Dakota::ProblemDescDB and from which all parameters are retrieved by the created algorithms.

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.

Friends

- class JEGAEvaluator

  The JEGAEvaluator is a friend so that it can access the methods of the Minimizer base class of the JEGAOptimizer necessary to do variable scaling.

Classes

- class EvalCreator

  A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a JEGAEvaluator.

- class JEGAProbDescDB

  A specialization of the JEGA::Utilities::ParameterDatabase that wraps and retrieves data from a Dakota::ProblemDescDB.

10.51.1 Detailed Description

Version of Optimizer for instantiation of John Eddy's Genetic Algorithms.

This class encapsulates the necessary functionality for creating and properly initializing a GeneticAlgorithm.

10.51.2 Constructor & Destructor Documentation

10.51.2.1 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.

10.51.3 Member Function Documentation

10.51.3.1 void ReCreateTheAlgorithmConfig () `[protected]`

Destroys the existing algorithm configuration and creates a new one.

A usable evaluator creator must already exist prior to calling this method. The parameter database will be destroyed and re-created in order to implement this method. As a result, the parameter data in the underlying ProblemDescDB will remain unchanged but any additional data will be gone.

10.51.3.2 void ReCreateTheProblemConfig () `[protected]`

Destroys the existing problem configuration and creates a new one.

The newly created problem configuration will be completely empty.

10.51.3.3 void LoadTheAlgorithmConfig () `[protected]`

Completely initializes or re-initializes the algorithm configuration.

This first uses the ReCreateTheAlgorithmConfig method and then loads the fresh configuration object with appropriate data retrieved from the parameter database.

10.51.3.4 void LoadTheProblemConfig () `[protected]`

Completely initializes or re-initializes the problem configuration.

This first uses the ReCreateTheProblemConfig method and then loads the fresh configuration object using the LoadTheDesignVariables, LoadTheObjectiveFunctions, and LoadTheConstraints methods.

10.51.3.5 void LoadTheDesignVariables () `[protected]`

Adds DesignVariableInfo objects into the problem configuration object.

This retrieves design variable information from the ParameterDatabase and creates DesignVariableInfo's from it.

10.51.3.6 void LoadTheObjectiveFunctions () `[protected]`

Adds ObjectiveFunctionInfo objects into the problem configuration object.

This retrieves objective function information from the ParameterDatabase and creates ObjectiveFunctionInfo's from it.

10.51.3.7 void LoadTheConstraints () `[protected]`

Adds ConstraintInfo objects into the problem configuration object.

This retrieves constraint function information from the ParameterDatabase and creates ConstraintInfo's from it.

10.51.3.8 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.

10.51.3.9 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.

10.51.3.10 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.

10.51.3.11 void find_optimum () `[virtual]`

Performs the iterations to determine the optimal set of solutions.

Override of pure virtual method in Optimizer base class.

Implements Optimizer.

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

- JEGAOptimizer.H
- JEGAOptimizer.C

10.52 JEGAOptimizer::EvalCreator Class Reference

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a JEGAEvaluator.

Public Member Functions

- virtual JEGA::Algorithms::GeneticAlgorithmEvaluator ∗ CreateEvaluator (JEGA::Algorithms::Genetic-Algorithm &alg)

  Overriden to return a newly created JEGAEvaluator.

- EvalCreator (Model &theModel, JEGAOptimizer &theOptimizer)

  Constructs an EvalCreator using the supplied model and optimizer.

Private Attributes

- JEGAOptimizer & _theOptimizer

  The optimizer instance to be passed to the constructor of the JEGAEvaluator.

- Model & _theModel

  The user defined model to be passed to the constructor of the JEGAEvaluator.

10.52.1 Detailed Description

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a JEGAEvaluator.

10.52.2 Constructor & Destructor Documentation

10.52.2.1 EvalCreator (Model & *theModel*, JEGAOptimizer & *theOptimizer*)

Constructs an EvalCreator 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.

10.52.3 Member Function Documentation

10.52.3.1 virtual JEGA::Algorithms::GeneticAlgorithmEvaluator$*$ CreateEvaluator (JEGA::Algorithms::GeneticAlgorithm & *alg*) `[virtual]`

Overriden to return a newly created JEGAEvaluator.

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 JEGAEvaluator are stored as members of this class at construction time.

Parameters:

alg The GA for which the evaluator is to be created.

Returns:

A pointer to a newly created Evaluator.

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

- JEGAOptimizer.C

10.53 JEGAOptimizer::JEGAProbDescDB Class Reference

A specialization of the JEGA::Utilities::ParameterDatabase that wraps and retrieves data from a Dakota::ProblemDescDB.

Public Member Functions

- virtual int GetIntegral (const std::string &tag) const

  Overridden to supply the requested parameter as an integer from this DB.

- virtual double GetDouble (const std::string &tag) const

  Overridden to supply the requested parameter as a double from this DB.

- virtual std::size_t GetSizeType (const std::string &tag) const

  Overridden to supply the requested parameter as a size_t from this DB.

- virtual bool GetBoolean (const std::string &tag) const

  Overridden to supply the requested parameter as a bool from this DB.

- virtual std::string GetString (const std::string &tag) const

  Overridden to supply the requested parameter as a string from this DB.

- virtual JEGA::DoubleVector GetDoubleVector (const std::string &tag) const

  Overridden to supply the requested parameter as a DoubleVector from this DB.

- virtual JEGA::IntVector GetIntVector (const std::string &tag) const

  Overridden to supply the requested parameter as an IntVector from this DB.

- virtual JEGA::DoubleMatrix GetDoubleMatrix (const std::string &tag) const

  Overridden to supply the requested parameter as a DoubleMatrix from this DB.

- virtual JEGA::IntList GetIntList (const std::string &tag) const

  Overridden to supply the requested parameter as an IntList from this DB.

- virtual JEGA::StringVector GetStringVector (const std::string &tag) const

  Overridden to supply the requested parameter as a StringVector from this DB.

- virtual JEGA::StringList GetStringList (const std::string &tag) const

  Overridden to supply the requested parameter as a StringList from this DB.

- virtual std::string Dump () const

  Prints the contents of the entire database into a string and return it.

- virtual void Dump (std::ostream &stream) const

  Prints the contents of the entire database into the supplied output stream.

- JEGAProbDescDB (const ProblemDescDB &wrapped)

  Constructs a JEGAProbDescDB to wrap wrapped.

Private Attributes

- const ProblemDescDB & _wrapped

  The Dakota::ProblemDescription database from which the actual data is obtained.

10.53.1 Detailed Description

A specialization of the JEGA::Utilities::ParameterDatabase that wraps and retrieves data from a Dakota::ProblemDescDB.

10.53.2 Constructor & Destructor Documentation

10.53.2.1 JEGAProbDescDB (const ProblemDescDB & *wrapped*)

Constructs a JEGAProbDescDB to wrap *wrapped*.

Parameters:

wrapped The existing ProblemDescDB from which otherwise unknown parameter values are retrieved.

10.53.3 Member Function Documentation

10.53.3.1 int GetIntegral (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as an integer from this DB.

Parameters:

tag The key by which the requested value is to be retrieved.

10.53.3.2 double GetDouble (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a double from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.3 size_t GetSizeType (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a size_t from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.4 bool GetBoolean (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a bool from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.5 string GetString (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a string from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.6 JEGA::DoubleVector GetDoubleVector (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a DoubleVector from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.7 JEGA::IntVector GetIntVector (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as an IntVector from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.8 JEGA::DoubleMatrix GetDoubleMatrix (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a DoubleMatrix from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.9 JEGA::IntList GetIntList (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as an IntList from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.10 JEGA::StringVector GetStringVector (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a StringVector from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.11 JEGA::StringList GetStringList (const std::string & *tag*) const `[virtual]`

Overridden to supply the requested parameter as a StringList from this DB.

Parameters:
tag The key by which the requested value is to be retrieved.

10.53.3.12 string Dump () const `[virtual]`

Prints the contents of the entire database into a string and return it.

This method cannot be implemented by this class and thus does nothing.

Returns:

The entire contents of the database in a string.

10.53.3.13 void Dump (std::ostream & *stream*) const `[virtual]`

Prints the contents of the entire database into the supplied output stream.

This method cannot be implemented by this class and thus does nothing.

Parameters:

stream The stream into which to write the contents of this database.

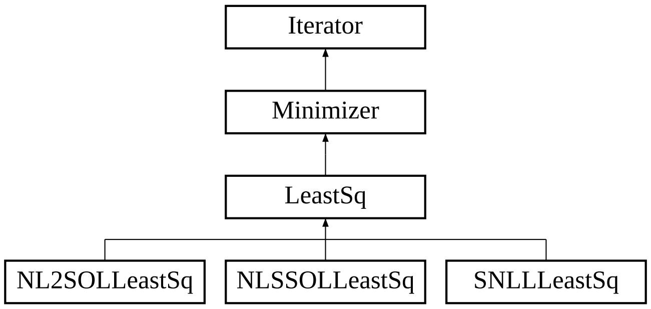
The documentation for this class was generated from the following file:

- JEGAOptimizer.C

10.54 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 ()

  provides derived class-specific portions of scaling initialization since Optimizer and LeastSq iterators have obj fn. and residual scales, respectively

- virtual void minimize_residuals ()=0

  Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run_iterator virtual function for the least squares branch.

Protected Attributes

- int numLeastSqTerms

  number of least squares terms

10.54.1 Detailed Description

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

The LeastSq class provides common data and functionality for NLSSOLLeastSq and SNLLLeastSq.

10.54.2 Constructor & Destructor Documentation

10.54.2.1 LeastSq (Model & *model*) `[protected]`

standard constructor

This constructor extracts the inherited data for the least squares branch and performs sanity checking on gradient and constraint settings.

10.54.3 Member Function Documentation

10.54.3.1 void run () `[inline, protected, 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.

10.54.3.2 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.

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

- DakotaLeastSq.H
- DakotaLeastSq.C

10.55 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.

- 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

  Returns TRUE if the list contains an object that the user defined function finds and sets k to this object.

- List< T >::iterator find (bool(∗test_fn)(const T &, const void ∗), const void ∗test_fn_data)

  Returns an iterator pointing to an object that the user defined 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.

- void sort (bool(∗sort_fn)(const T &, const T &))

  Sorts the list into an order based on the predefined sort function.

- 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).

10.55.1 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

10.55.2 Member Function Documentation

10.55.2.1 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.

10.55.2.2 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.

10.55.2.3 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.

10.55.2.4 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.

10.55.2.5 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.

10.55.2.6 bool find (bool(∗)(const T &, const void ∗) *test_fn*, const void ∗ *test_fn_data*, T & *found_item*) const

Returns TRUE if the list contains an object that the user defined 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.

10.55.2.7 List< T >::iterator find (bool(∗)(const T &, const void ∗) *test_fn*, const void ∗ *test_fn_data*)

Returns an iterator pointing to an object that the user defined function finds.

Find the first item in the list which satisfies the test function and return an iterator pointing to it.

10.55.2.8 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).

10.55.2.9 void sort (bool(∗)(const T &, const T &) *sort_fn*) `[inline]`

Sorts the list into an order based on the predefined sort function.

The sort method utilizes the SortCompare functor and the base class list::sort algorithm to sort a list based on the incoming sorting function sort_fn. Note that the functor-based sorting method of std::list is not supported by all compilers (e.g., SOLARIS, TFLOP) due to use of member templates, but a function pointer-based interface is available in some cases.

10.55.2.10 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).

10.55.2.11 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 occurences of the specified object.

10.55.2.12]

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->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.

10.55.2.13]

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->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.

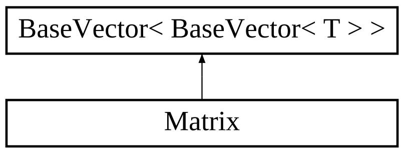
The documentation for this class was generated from the following file:

- DakotaList.H

10.56 Matrix Class Template Reference

Template class for the Dakota numerical matrix.

Inheritance diagram for 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.

- 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.

10.56.1 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.).

10.56.2 Member Function Documentation

10.56.2.1 Matrix< T > & operator= (const T & *val*) `[inline]`

Sets all elements in the matrix to ival.

calls base class operator=(ival)

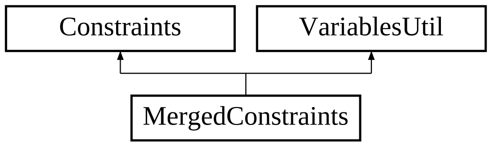
The documentation for this class was generated from the following file:

- DakotaMatrix.H

10.57 MergedConstraints Class Reference

Derived class within the Constraints hierarchy which employs the merged data view.

Inheritance diagram for 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

- 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

- RealVector all_continuous_upper_bounds () const

  returns a single array with all continuous upper bounds

- 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

Private Attributes

- RealVector mergedDesignLowerBnds

  a design lower bounds array merging continuous and discrete domains (integer values promoted to reals)

- RealVector mergedDesignUpperBnds

  a design upper bounds array merging continuous and discrete domains (integer values promoted to reals)

- RealVector uncertainLowerBnds

  the uncertain distribution lower bounds array (no discrete uncertain to merge)

- RealVector uncertainUpperBnds

  the uncertain distribution upper bounds array (no discrete uncertain to merge)

- RealVector mergedStateLowerBnds

  a state lower bounds array merging continuous and discrete domains (integer values promoted to reals)

- RealVector mergedStateUpperBnds

  a state upper bounds array merging continuous and discrete domains (integer values promoted to reals)

10.57.1 Detailed Description

Derived class within the Constraints hierarchy which employs 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 (uncertain-LowerBnds, uncertainUpperBnds), and merged state bounds arrays (mergedStateLowerBnds, mergedStateUpper-Bnds). 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).

10.57.2 Constructor & Destructor Documentation

10.57.2.1 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, merged-DesignUpperBnds, mergedStateLowerBnds, and mergedStateUpperBnds using utilities from VariablesUtil (un-certain distribution bounds do not require any merging).

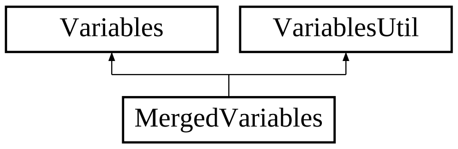
The documentation for this class was generated from the following files:

- MergedConstraints.H
- MergedConstraints.C

10.58 MergedVariables Class Reference

Derived class within the Variables hierarchy which employs the merged data view.

Inheritance diagram for 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 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

- 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

Private Member Functions

- void copy_rep (const Variables ∗vars_rep)

  Used by copy() to copy the contents of a letter class.

Private Attributes

- RealVector mergedDesignVars

  a design variables array merging continuous and discrete domains (integer values promoted to reals)

- RealVector uncertainVars

  the uncertain variables array (no discrete uncertain to merge)

- RealVector mergedStateVars

  a state variables array merging continuous and discrete domains (integer values promoted to reals)

- StringArray mergedDesignLabels

  a label array combining continuous design and discrete design labels

- StringArray uncertainLabels

  the uncertain variables label array (no discrete uncertain to combine)

- StringArray mergedStateLabels

  a label array combining continuous state and discrete state labels

Friends

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

  equality operator

10.58.1 Detailed Description

Derived class within the Variables hierarchy which employs the 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)).

10.58.2 Constructor & Destructor Documentation

10.58.2.1 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, mergedState-Vars, mergedDesignLabels, and mergedStateLabels using utilities from VariablesUtil (uncertain variables and labels do not require any merging).

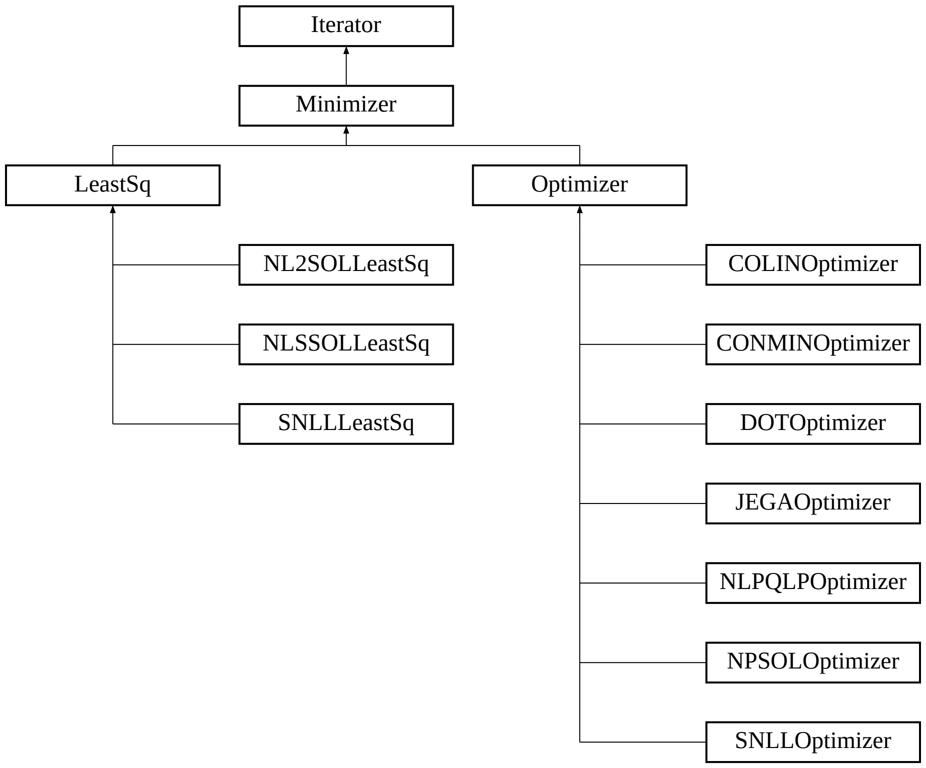
The documentation for this class was generated from the following files:

- MergedVariables.H
- MergedVariables.C

10.59 Minimizer Class Reference

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

Inheritance diagram for Minimizer::



Public Member Functions

- const Variables & variable_results () const

  return the final iterator solution (variables)

- const Response & response_results () const

  return the final iterator solution (response)

Protected Member Functions

- Minimizer ()

  default constructor

- Minimizer (Model &model)

  standard constructor

- 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 ()=0

  provides derived class-specific portions of scaling initialization since Optimizer and LeastSq iterators have obj fn. and residual scales, respectively

- void initialize_scaling ()

  helper function to initialize scaling multipliers and offets

- RealVector modify_n2s (const RealVector &native_vars, const RealVector &multipliers, const RealVector &offsets) const

  general mapping from native to scaled variables vectors: scaled_vars = (native_vars - offsets)/multiplierss

- RealVector cv_modify_n2s (const RealVector &native_vars) const

  map continuous variables from native to scaled

- RealVector nln_ineq_modify_n2s (const RealVector &native_vars) const

  map nonlinear inequalities from native to scaled

- RealVector nln_eq_modify_n2s (const RealVector &native_vars) const

  map nonlinear equalities from native to scaled

- RealVector lin_ineq_modify_n2s (const RealVector &native_vars) const

  map linear inequalities from native to scaled

- RealVector lin_eq_modify_n2s (const RealVector &native_vars) const

  map linear equalities from native to scaled

- RealVector fns_modify_n2s (const RealVector &native_fns) const

  map functions (reponse vector) from native to scaled

- 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

- RealMatrix lin_ineq_coeffs_modify_n2s (const RealMatrix &native_coeffs) const

  map linear inequality constraint matrix from native to scaled

- RealMatrix lin_eq_coeffs_modify_n2s (const RealMatrix &native_coeffs) const

  map linear inequality constraint matrix from native to scaled

- Response response_modify_n2s (const Response &response_source) const

  map reponses from native to scaled variable space

- RealVector modify_s2n (const RealVector &scaled_vars, const RealVector &multipliers, const RealVector &offsets) const

  general RealVector mapping from native to scaled variables: native_vars = scaled_vars ∗ multipliers + offsets

- RealVector cv_modify_s2n (const RealVector &scaled_vars) const

  map continuous variables from scaled to native

- RealVector fns_modify_s2n (const RealVector &scaled_fns) const

  map functions (responses) from scaled to native

- void adjust_user_scales (RealVector &scales, const int length, const Real default_scale_factor)

  expand and error check user-supplied scaling vectors before use in scaling framework

- void compute_scale_factor (const Real lower_bound, const Real upper_bound, Real ∗multiplier, Real ∗offset)

  automatically compute scaling factor – bounds case

- void compute_scale_factor (const Real target, Real ∗multiplier)

  automatically compute scaling factor – target case

Protected Attributes

- Real constraintTol

  optimizer/least squares constraint tolerance

- Real bigRealBoundSize

  cutoff value for inequality constraint and continuous variable bounds

- int bigIntBoundSize

  cutoff value for discrete variable bounds

- size_t numNonlinearIneqConstraints

  number of nonlinear inequality constraints

- size_t numNonlinearEqConstraints

number of nonlinear equality constraints

- size_t numLinearIneqConstraints

  number of linear inequality constraints

- size_t numLinearEqConstraints

  number of linear equality constraints

- int numNonlinearConstraints

  total number of nonlinear constraints

- int numLinearConstraints

  total number of linear constraints

- int numConstraints

  total number of linear and nonlinear constraints

- bool boundConstraintFlag

  convenience flag for denoting the presence of user-specified bound constraints. Used for method selection and error checking.

- bool speculativeFlag

  flag for speculative gradient evaluations

- bool scaleFlag

  flag indicating scaling status

- RealVector cvScaleMultipliers

  scales for continuous variables

- RealVector cvScaleOffsets

  offsets for continuous variables

- RealVector fnScaleMultipliers

  scales for obj. fns. or LSQ terms

- RealVector nonlinearIneqScaleMultipliers

  scales for nonlin. ineq.

- RealVector nonlinearIneqScaleOffsets

  offsets for nonlin. ineq.

- RealVector nonlinearEqScaleMultipliers

  scales for nonlin. eq.

- RealVector nonlinearEqScaleOffsets

offsets for nonlin. eq.

- RealVector responseScaleMultipliers

  scales for ALL responses

- RealVector responseScaleOffsets

  offsets for ALL responses (zero `for` *functions, not for nonlin con)*

- RealVector linearIneqScaleMultipliers

  scales for linear ineq constrs.

- RealVector linearIneqScaleOffsets

  offsets for linear ineq constrs.

- 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 solution

- Response bestResponses

  best responses found in solution

Friends

- class SOLBase

  the SOLBase class is not derived the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)

- class SNLLBase

  the SNLLBase class is not derived the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)

10.59.1 Detailed Description

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

The Minimizer class provides common data and functionality for Optimizer and LeastSq.

10.59.2 Constructor & Destructor Documentation

10.59.2.1 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.

10.59.3 Member Function Documentation

10.59.3.1 void initialize_scaling () `[protected]`

helper function to initialize scaling multipliers and offets

helper function used in constructors of derived classes to set up scaling multipliers and offsets when input scaling flag is enabled includes call to the derived class' derived_initialize_scaling()

10.59.3.2 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

10.59.3.3 Response response_modify_n2s (const Response & *src_response*) const `[protected]`

map reponses from native to scaled variable space

scaling forward mapping: modifies response from a model (native) for use in iterators (scaled) with scaling and multi_objective or building least squares terms – not including multi_obejctive_modify, since least squares methods do not use

10.59.3.4 void adjust_user_scales (RealVector & *scales*, const int *length*, const Real *default_scale_factor*) `[protected]`

expand and error check user-supplied scaling vectors before use in scaling framework

expand and error check user-specified scales vector if no scales: populate with default_scale_factor if only one scale factor: replicate to appropriate length

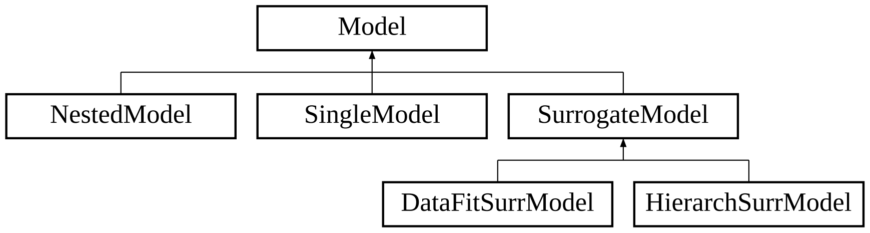
The documentation for this class was generated from the following files:

- DakotaMinimizer.H
- DakotaMinimizer.C

10.60 Model Class Reference

Base class for the model class hierarchy.

Inheritance diagram for Model::



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 Interface & interface ()

  return the interface employed by the derived model class, if present: SingleModel::userDefinedInterface, DataFitSurrModel::approxInterface, or NestedModel::optionalInterface

- virtual void surrogate_bypass (bool bypass_flag)

  deactivate/reactivate the approximations for any/all surrogate models contained within this model

- virtual void build_approximation ()

  build a new approximation in SurrogateModels

- virtual bool build_approximation (const RealVector &c_vars, const Response &response)

  build a new approximation in SurrogateModels using/enforcing response at c_vars

- virtual void update_approximation (const RealVector &c_vars, const Response &response)

  update an existing approximation in DataFitSurrModels with new data

- virtual const RealVectorArray & approximation_coefficients ()

  retrieve the approximation coefficients from each Approximation within a DataFitSurrModel

- 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 ()

  return flag indicating use of automatic correction within this 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 (int mode)

  update component parallel mode for supporting parallelism in a model's interface component, sub-model component, or neither component [componentParallelMode = 0 (none), 1 (INTERFACE/LF_MODEL), or 2 (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 reset_communicators ()

reset communicator partition data for a model

- virtual void serve ()

  Service job requests received from the master. Completes when a termination message is received from stop_servers().

- virtual void stop_servers ()

  Executed by the master to terminate all server operations for a particular model when iteration on the model is complete.

- virtual bool derived_master_overload () const

  Return a flag indicating the combination of multiprocessor evaluations and a dedicated master iterator scheduling. Used in synchronous compute_response functions to prevent the error of trying to run a multiprocessor job on the master.

- virtual 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 ()

  Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (default ActiveSet).

- void asynch_compute_response (const ActiveSet &set)

  Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (specified ActiveSet).

- const ResponseArray & synchronize ()

  Execute a blocking scheduling algorithm to collect the complete set of results from a group of asynchronous evaluations.

- const IntResponseMap & synchronize_nowait ()

Execute a nonblocking scheduling algorithm to collect all available results from a group of asynchronous evaluations.

- void init_communicators (const int &max_iterator_concurrency)

  allocate communicator partitions for a model and store configuration in modelPCIterMap

- void init_serial ()

  for cases where init_communicators() will not be called, modify some default settings to behave properly in serial.

- void set_communicators (const int &max_iterator_concurrency)

  set active parallel configuration for the model (set modelPCIter from modelPCIterMap)

- void free_communicators (const int &max_iterator_concurrency)

  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 num_functions () const

  return number of functions in currentResponse

- 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

- 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

- size_t icv () const

  return number of inactive continuous variables

- size_t idv () const

  return number of inactive discrete variables

- 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

- 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 & 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 IntVector & interval_num_intervals () const

    return the interval number of intervals per variable

- void interval_num_intervals (const IntVector &int_num_intvls)

    set the interval number of intervals per variable

- const RealVector & interval_probs () const

    return the interval probability values

- void interval_probs (const RealVector &int_probs)

    set the interval probability values

- const RealVector & interval_bounds () const

    return the interval bounds

- void interval_bounds (const RealVector &int_bounds)

    set the interval probability values

- 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

- const RealVector & continuous_lower_bounds () const

  return the active continuous variable lower bounds from userDefinedConstraints

- void continuous_lower_bounds (const RealVector &c_l_bnds)

  set the active continuous variable lower bounds in userDefinedConstraints

- const RealVector & continuous_upper_bounds () const

  return the active continuous variable upper bounds from userDefinedConstraints

- void continuous_upper_bounds (const RealVector &c_u_bnds)

  set the active continuous variable upper bounds in userDefinedConstraints

- const IntVector & discrete_lower_bounds () const

  return the active discrete variable lower bounds from userDefinedConstraints

- void discrete_lower_bounds (const IntVector &d_l_bnds)

  set the active discrete variable lower bounds in userDefinedConstraints

- const IntVector & discrete_upper_bounds () const

  return the active discrete variable upper bounds from userDefinedConstraints

- void discrete_upper_bounds (const IntVector &d_u_bnds)

  set the active discrete variable 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

- 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 IntList & merged_integer_list () const
  return the list of discrete variables merged into a continuous array in currentVariables
- const IntArray & message_lengths () const
  return the array of MPI packed message buffer lengths (messageLengths)
- const Variables & current_variables () const
  return the current variables (currentVariables)
- const Response & current_response () const
  return the current response (currentResponse)

- const ProblemDescDB & prob_desc_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)

- const String & interface_id ()

  return the interface identifier (interface().interface_id())

- bool asynch_flag () const

  return the asynchronous evaluation flag (asynchEvalFlag)

- void asynch_flag (const bool flag)

  set the asynchronous evaluation flag (asynchEvalFlag)

- void auto_graphics (const bool flag)

  set modelAutoGraphicsFlag to activate posting of graphics data within compute_response/synchronize functions (automatic graphics posting in the model as opposed to graphics posting at the strategy level).

- const String & gradient_method () const

  return the gradient evaluation method (gradType)

- const String & hessian_method () const

  return the Hessian evaluation method (hessType)

- 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

- void parallel_configuration_iterator (const ParConfigLIter &pc_iter)

  set modelPCIter

- const ParConfigLIter & parallel_configuration_iterator () const

  return modelPCIter

- bool is_null () const

  function to check modelRep (does this envelope contain a letter)

Protected Member Functions

- Model (BaseConstructor, ProblemDescDB &problem_db)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- virtual void 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)

  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)

  portion of set_communicators() specific to derived model classes

- virtual void derived_free_communicators (const int &max_iterator_concurrency)

  portion of free_communicators() specific to derived model classes

Protected Attributes

- Variables currentVariables

  the set of current variables used by the model for performing function evaluations

- size_t numDerivVars

  the number of active continuous variables used in computing most response derivatives (i.e., in places such as quasi-Hessians and response corrections where only the active continuous variables are supported)

- Response currentResponse

  the set of current responses that holds the results of model function evaluations

- size_t numFns

  the number of functions in currentResponse

- Constraints userDefinedConstraints

  Explicit constraints on variables are maintained in the Constraints class hierarchy. Currently, this includes linear constraints and bounds, but could be extended in the future to include other explicit constraints which (1) have their form specified by the user, and (2) are not catalogued in Response since their form and coefficients are published to an iterator at startup.

- IntArray messageLengths

  length of packed MPI buffers containing vars, vars/set, response, and PRPair

- const ProblemDescDB & probDescDB

  class member reference to the problem description database. This reference is a const copy of the incoming problem_db non-const reference and is only used in Model::prob_desc_db() (it is not inherited).

- ParallelLibrary & parallelLib

  class member reference to the parallel library

- ParConfigLIter modelPCIter

  the ParallelConfiguration node used by this model instance

- int componentParallelMode

  the component parallelism mode: 0 (none), 1 (INTERFACE/LF_MODEL), or 2 (SUB_MODEL/HF_-MODEL/TRUTH_MODEL)

Private Member Functions

- Model ∗ get_model (ProblemDescDB &problem_db)

  Used by the envelope to instantiate the correct letter class.

- int estimate_derivatives (const IntArray &map_asv, const IntArray &fd_grad_asv, const IntArray &fd_-hess_asv, const IntArray &quasi_hess_asv, const ActiveSet &original_set, const bool asynch_flag)

  evaluate numerical gradients using finite differences. This routine is selected with "method_source dakota" (the default method_source) in the numerical gradient specification.

- void synchronize_derivatives (const Variables &vars, const ResponseArray &fd_responses, Response &new_response, const IntArray &fd_grad_asv, const IntArray &fd_hess_asv, const IntArray &quasi_-hess_asv, const ActiveSet &original_set)

  combine results from an array of finite difference response objects (fd_grad_responses) into a single response (new_response)

- void update_response (const Variables &vars, Response &new_response, const IntArray &fd_grad_asv, const IntArray &fd_hess_asv, const IntArray &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 IntArray &asv_in, IntArray &map_asv_out, IntArray &fd_grad_asv_out, IntArray &fd_hess_asv_out, IntArray &quasi_hess_asv_out)

  Coordinates usage of estimate_derivatives() calls based on asv_in.

Private Attributes

- Model ∗ modelRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing modelRep

- String modelType

  type of model: single, nested, or surrogate

- bool hierarchicalModel

  flag for identifying a HierarchSurrModel

- String idModel

  model identifier string from the input file

- bool estDerivsFlag

  flags presence of estimated derivatives within a set of calls to asynch_compute_response()

- bool asynchEvalFlag

  flags asynch evaluations (local or distributed)

- int evaluationCapacity

  capacity for concurrent evaluations supported by the Model

- std::map< int, ParConfigLIter > modelPCIterMap

  map<> used for tracking modelPCIter instances using concurrency level as the lookup key

- bool modelAutoGraphicsFlag

  flag for posting of graphics data within compute_response (automatic graphics posting in the model as opposed to graphics posting at the strategy level)

- 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

  history of vars populated in asynch_compute_response() and used in synchronize().

- List< IntArray > asvList

  if estimate_derivatives() is used, transfers ASVs from asynch_compute_response() to synchronize()

- List< ActiveSet > setList

  if estimate_derivatives() is used, transfers ActiveSets from asynch_compute_response() to synchronize()

- BoolList initialMapList

  transfers initial_map flag values from estimate_derivatives() to synchronize_derivatives()

- BoolList dbCaptureList

  transfers db_capture flag values from estimate_derivatives() to synchronize_derivatives()

- ResponseList dbResponseList

  transfers database captures from estimate_derivatives() to synchronize_derivatives()

- RealList deltaList

  transfers deltas from estimate_derivatives() to synchronize_derivatives()

- IntList numMapsList

  tracks the number of maps used in estimate_derivatives(). Used in synchronize() as a key for combining finite difference responses 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

  used to return an array of responses for asynchronous evaluations. This array has the responses in final concatenated form. The similar array in Interface contains the raw responses.

- IntResponseMap graphicsRespMap

  used to cache the data returned from derived_synchronize_nowait() prior to sequential input into the graphics

- 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

- 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 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)

- IntVector intervalNumIntervals

  interval uncertain variable number of intervals per variable

- RealVector intervalProbValues

  interval uncertain variable probability values

- RealVector intervalBounds

  interval uncertain variable interval bounds

10.60.1 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.

10.60.2 Constructor & Destructor Documentation

10.60.2.1 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.

10.60.2.2 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.

10.60.2.3 Model (const Model & *model*)

copy constructor

Copy constructor manages sharing of modelRep and incrementing of referenceCount.

10.60.2.4 ~Model () `[virtual]`

destructor

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

10.60.2.5 Model (BaseConstructor, ProblemDescDB & *problem_db*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor 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).

10.60.3 Member Function Documentation

10.60.3.1 Model operator= (const Model & *model*)

assignment operator

Assignment operator decrements referenceCount for old modelRep, assigns new modelRep, and increments referenceCount for new modelRep.

10.60.3.2 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.

10.60.3.3 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.

10.60.3.4 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.

10.60.3.5 Interface & interface () `[virtual]`

return the interface employed by the derived model class, if present: SingleModel::userDefinedInterface, DataFitSurrModel::approxInterface, or NestedModel::optionalInterface return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in DataFitSurrModel, NestedModel, and SingleModel.

10.60.3.6 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 SingleModel.

10.60.3.7 int local_eval_concurrency () `[virtual]`

return derived model asynchronous evaluation concurrency

SingleModels and HierarchSurrModels redefine this virtual function.

Reimplemented in SingleModel.

10.60.3.8 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).

10.60.3.9 void init_communicators (const int & *max_iterator_concurrency*)

allocate communicator partitions for a model and store configuration in modelPCIterMap

The init_communicators() and derived_init_communicators() functions are stuctured 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.

10.60.3.10 void init_serial ()

for cases where init_communicators() will not be called, modify some default settings to behave properly in serial.

The init_serial() and derived_init_serial() functions are stuctured to separate base class (common) operations from derived class (specialized) operations.

10.60.3.11 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.

10.60.3.12 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).

10.60.3.13 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.

10.60.3.14 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.

10.60.3.15 int estimate_derivatives (const IntArray & *map_asv*, const IntArray & *fd_grad_asv*, const IntArray & *fd_hess_asv*, const IntArray & *quasi_hess_asv*, const ActiveSet & *original_set*, const bool *asynch_flag*) `[private]`

evaluate numerical gradients using finite differences. This routine is selected with "method_source dakota" (the default method_source) in the numerical gradient specification.

Estimate derivatives by computing finite difference gradients, finite difference Hessians, and/or quasi-Newton Hessians. The total number of finite difference evaluations is returned for use by synchronize() to track response arrays, and it could be used to improve management of max_function_evaluations within the iterators.

10.60.3.16 void synchronize_derivatives (const Variables & *vars*, const ResponseArray & *fd_responses*, Response & *new_response*, const IntArray & *fd_grad_asv*, const IntArray & *fd_hess_asv*, const IntArray & *quasi_hess_asv*, const ActiveSet & *original_set*) `[private]`

combine results from an array of finite difference response objects (fd_grad_responses) into a single response (new_response)

Merge an array of fd_responses into a single new_response. This function is used both by synchronous compute_response() for the case of asynchronous estimate_derivatives() and by synchronize() for the case where one or more asynch_compute_response() calls has employed asynchronous estimate_derivatives().

10.60.3.17 void update_response (const Variables & *vars*, Response & *new_response*, const IntArray & *fd_grad_asv*, const IntArray & *fd_hess_asv*, const IntArray & *quasi_hess_asv*, const ActiveSet & *original_set*, Response & *initial_map_response*, const RealMatrix & *new_fn_grads*, 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.

10.60.3.18 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.

10.60.3.19 bool manage_asv (const IntArray & *asv_in*, IntArray & *map_asv_out*, IntArray & *fd_grad_asv_out*, IntArray & *fd_hess_asv_out*, IntArray & *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

10.61 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 ()

  Desctructor.

- 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.

10.61.1 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

10.62 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 **fbat'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 **fbat***.*

- 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.*

10.62.1 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]

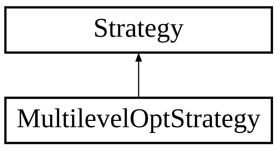
The documentation for this class was generated from the following files:

- MPIPackBuffer.H
- MPIPackBuffer.C

10.63 MultilevelOptStrategy Class Reference

Strategy for hybrid optimization using multiple optimizers on multiple models of varying fidelity.

Inheritance diagram for MultilevelOptStrategy::



Public Member Functions

- MultilevelOptStrategy (ProblemDescDB &problem_db)
  constructor
- ∼MultilevelOptStrategy ()
  destructor
- void run_strategy ()
  Performs the hybrid optimization strategy by executing multiple iterators on different models of varying fidelity.
- const Variables & variable_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

- Real localSearchProb

  the probability of running a local search refinement within phases of the global optimization for coupled hybrids

- Real progressMetric

  the amount of progress made in a single iterator++ cycle within an uncoupled adaptive hybrid

- Real progressThreshold

  when the progress metric falls below this threshold, the 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

10.63.1 Detailed Description

Strategy for hybrid optimization using multiple optimizers on 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 adapatively 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.

10.63.2 Member Function Documentation

10.63.2.1 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.

10.63.2.2 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.

10.63.2.3 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.

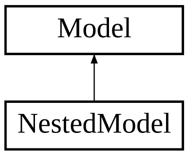
The documentation for this class was generated from the following files:

- MultilevelOptStrategy.H
- MultilevelOptStrategy.C

10.64 NestedModel Class Reference

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

Inheritance diagram for NestedModel::



Public Member Functions

- NestedModel (ProblemDescDB &problem_db)

  constructor

- ∼NestedModel ()

  destructor

Protected Member Functions

- void derived_compute_response (const ActiveSet &set)

  portion of compute_response() specific to NestedModel

- void derived_asynch_compute_response (const ActiveSet &set)

  portion of asynch_compute_response() specific to NestedModel

- void derived_subordinate_models (ModelList &ml, bool recurse_flag)

  return subModel

- Iterator & subordinate_iterator ()

  return subIterator

- Interface & interface ()

  return optionalInterface

- void surrogate_bypass (bool bypass_flag)

  NestedModels have nothing to bypass, but must pass request on to the subModel for any lower-level surrogates.

- void component_parallel_mode (int mode) *update component parallel mode for supporting parallelism in optionalInterface and subModel*

- bool derived_master_overload () const

  flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)

- void derived_init_communicators (const int &max_iterator_concurrency)

  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)

  set active parallel configuration within subModel

- void reset_communicators ()

  reset communicator partitions for the NestedModel (forwarded to optionalInterface and subModel)

- void derived_free_communicators (const int &max_iterator_concurrency)

  deallocate communicator partitions for the NestedModel (forwarded to optionalInterface and subModel)

- void serve ()

  Service optionalInterface and subModel job requests received from the master. Completes when a termination message is received from stop_servers().

- void stop_servers ()

  Executed by the master to terminate server operations for subModel and optionalInterface when iteration on the NestedModel is complete.

- int evaluation_id () const

  Return the current evaluation id for the NestedModel.

- void set_evaluation_reference ()

  set the evaluation counter reference points for the NestedModel (request forwarded to optionalInterface and subModel)

- void print_evaluation_summary (ostream &s, bool minimal_header=false, bool relative_count=true) const

  print the evaluation summary for the NestedModel (request forwarded to optionalInterface and subModel)

Private Member Functions

- void asv_mapping (const IntArray &mapped_asv, IntArray &interface_asv, IntArray &sub_iterator_asv)

  define the evaluation requirements for the optionalInterface (interface_asv) and the subIterator (sub_iterator_asv) from the total model evaluation requirements (mapped_asv)

- void response_mapping (const Response &interface_response, const Response &sub_iterator_response, Response &mapped_response)

  combine the response from the optional interface evaluation with the response from the sub-iteration using the primaryCoeffs/secondaryCoeffs mappings to create the total response for the model

- void update_sub_model ()

  update subModel with current variable values/bounds/labels

Private Attributes

- int nestedModelEvals

  number of calls to derived_compute_response()/ 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

  number of top-level inequality constraints mapped from the sub-iteration results

- size_t numSubIterMappedEqCon

  number of top-level equality constraints mapped from the sub-iteration results

- Interface optionalInterface

  the optional interface contributes nonnested response data to the total model response

- String optInterfacePointer

  the optional interface pointer from the nested model specification

- Response optInterfaceResponse

  the response object resulting from optional interface evaluations

- size_t numOptInterfPrimary

  number of primary response functions (objective/least squares/generic functions) resulting from optional interface evaluations

- size_t numOptInterfIneqCon

  number of inequality constraints resulting from optional interface evaluations

- size_t numOptInterfEqCon

number of equality constraints resulting from the optional interface evaluations

- SizetArray primaryCVarMapIndices

  "primary" variable mappings for inserting active continuous currentVariables into active continuous subModel variables. If there are no secondary mappings defined, then the insertions replace the subModel variable values.

- SizetArray primaryDVarMapIndices

  "primary" variable mappings for inserting active discrete currentVariables into active discrete subModel variables. No secondary mappings are defined for discrete variables, so the insertions replace the subModel variable values.

- SizetArray secondaryVarMapIndices

  "secondary" variable mappings for inserting active continuous currentVariables into sub-parameters (e.g., distribution parameters for uncertain variables) of the active continuous subModel variables.

- RealMatrix primaryRespCoeffs

  "primary" response_mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level objective functions/least squares/ generic response terms.

- RealMatrix secondaryRespCoeffs

  "secondary" response_mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level inequality and equality constraints.

10.64.1 Detailed Description

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

The NestedModel class nests a sub-iterator execution within every model evaluation. This capability is most commonly used for optimization under uncertainty, in which a nondeterministic iterator is executed on every optimization function evaluation. The NestedModel also contains an optional interface, for portions of the model evaluation which are independent from the sub-iterator, and a set of mappings for combining sub-iterator and optional interface data into a top level response for the model.

10.64.2 Member Function Documentation

10.64.2.1 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.

10.64.2.2 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.

10.64.2.3 bool derived_master_overload () const `[inline, protected, virtual]`

flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)

Derived master overload for subModel is handled separately in subModel.compute_response() within subIterator.run().

Reimplemented from Model.

10.64.2.4 void derived_init_communicators (const int & *max_iterator_concurrency*) `[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.

10.64.2.5 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.

10.64.2.6 void response_mapping (const Response & *opt_interface_response*, const Response & *sub_iterator_response*, Response & *mapped_response*) `[private]`

combine the response from the optional interface evaluation with the response from the sub-iteration using the primaryCoeffs/secondaryCoeffs mappings to create the total response for the model

In the OUU case,

```
optionalInterface fns = {f}, {g} (deterministic primary functions, constraints)
subIterator fns       = {S}      (UQ response statistics)

Problem formulation for mapped functions:
                  minimize    {f} + [W]{S}
                  subject to  {g_l} <= {g}    <= {g_u}
                              {a_l} <= [A]{S} <= {a_u}
                              {g}    == {g_t}
                              [A]{S} == {a_t}
```

where [W] is the primary_mapping_matrix user input (primaryRespCoeffs class attribute), [A] is the secondary_-mapping_matrix user input (secondaryRespCoeffs class attribute), {{g_l},{a_l}} are the top level inequality constraint lower bounds, {{g_u},{a_u}} are the top level inequality constraint upper bounds, and {{g_t},{a_t}} are the top level equality constraint targets.

NOTE: optionalInterface/subIterator primary fns (obj/lsq/generic fns) overlap but optionalInterface/subIterator secondary fns (ineq/eq constraints) do not. The [W] matrix can be specified so as to allow

- some purely deterministic primary functions and some combined: [W] filled and [W].num_rows() < {f}.length() [combined first] *or* [W].num_rows() == {f}.length() and [W] contains rows of zeros [combined last]
- some combined and some purely stochastic primary functions: [W] filled and [W].num_rows() > {f}.length()
- separate deterministic and stochastic primary functions: [W].num_rows() > {f}.length() and [W] contains {f}.length() rows of zeros.

If the need arises, could change constraint definition to allow overlap as well: {g_l} <= {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.

10.64.3 Member Data Documentation

10.64.3.1 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

10.65 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)

10.65.1 Detailed Description

Auxiliary information passed to calcr and calcj via ur.

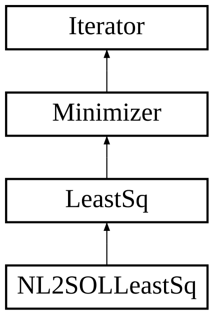
The documentation for this struct was generated from the following file:

- NL2SOLLeastSq.C

10.66 NL2SOLLeastSq Class Reference

Wrapper class for the NL2SOL nonlinear least squares library.

Inheritance diagram for NL2SOLLeastSq::



Public Member Functions

- NL2SOLLeastSq (Model &model)

  standard constructor

- ∼NL2SOLLeastSq ()

  destructor

- void minimize_residuals ()

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

  auxilary printing bits (see Dakota Ref Manual): sum of 1 = x0prt (print initial guess) 2 = solprt (print final solution) 4 = statpr (print solution statistics) 8 = parprt (print nondefault parameters) 16 = dradpr (print bound constraint drops/adds) debug/verbose/normal use default = 31 (everything), quiet uses 3, silent uses 0.

- int outlev

  frequency of output summary lines in number of iterations (debug/verbose/normal/quiet use default = 1, silent uses 0)

- Real dltfdj

  finite-diff step size for computing Jacobian approximation (`fd_gradient_step_size`*)*

- Real delta0

  finite-diff step size for gradient differences for H (a component of some covariance approximations, if desired) (`fd_hessian_step_size`*)*

- Real dltfdc

  finite-diff step size for function differences for H (`fd_hessian_step_size`*)*

- int mxfcal

  function-evaluation limit (`max_function_evaluations`*)*

- int mxiter

  iteration limit (`max_iterations`*)*

- Real rfctol

  relative fn convergence tolerance (`convergence_tolerance`*)*

- Real afctol

  absolute fn convergence tolerance (`absolute_conv_tol`*)*

- Real xctol

  x-convergence tolerance (`x_conv_tol`*)*

- Real sctol

  singular convergence tolerance (`singular_conv_tol`*)*

- Real lmaxs

  radius for singular-convergence test (`singular_radius`*)*

- Real xftol

  false-convergence tolerance (`false_conv_tol`*)*

- int covreq

  kind of covariance required (`covariance`*): 1 or -1 ==> sigma^2 H^-1 J^T J H^-1 2 or -2 ==> sigma^2 H^-1 3 or -3 ==> sigma^2 (J^T J)^-1 1 or 2 ==> use gradient diffs to estimate H -1 or -2 ==> use function diffs to estimate H default = 0 (no covariance)*

- 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

  pointer to the active object instance used within the static evaluator functions

10.66.1 Detailed Description

Wrapper class for the NL2SOL nonlinear least squares library.

The NL2SOLLeastSq class provides a wrapper for NL2SOL, a C library from Bell Labs. It uses a function pointer approach for which passed functions must be either global functions or static member functions.

10.66.2 Member Function Documentation

10.66.2.1 void minimize_residuals () `[virtual]`

Details on the following subscript values appear in "Usage Summary for Selected Optimization Routines" by David M. Gay, Computing Science Technical Report No. 153, AT&T Bell Laboratories, 1990. `http://netlib.bell-labs.com/cm/cs/cstr/153.ps.gz`

Implements LeastSq.

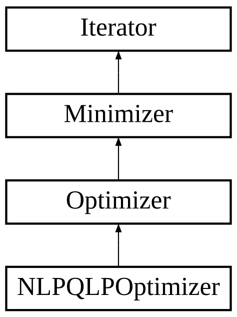
The documentation for this class was generated from the following files:

- NL2SOLLeastSq.H
- NL2SOLLeastSq.C

10.67 NLPQLPOptimizer Class Reference

Wrapper class for the NLPQLP optimization library, Version 2.0.

Inheritance diagram for NLPQLPOptimizer::



Public Member Functions

- NLPQLPOptimizer (Model &model)

    constructor

- ∼NLPQLPOptimizer ()

    destructor

- void find_optimum ()

    Used within the optimizer branch for computing the optimal solution. Redefines the run_iterator virtual function for the optimizer branch.

Protected Member Functions

- virtual void derived_pre_run ()

    performs run-time set up

- virtual void derived_post_run ()

    performs final solution processing

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

  L : Number of parallel systems, i.e. function calls during line search at predetermined iterates. HINT: If only less than 10 parallel function evaluations are possible, it is recommended to apply the serial version by setting L=1.

- int numEqConstraints

  numEqConstraints : Number of equality constraints.

- int MMAX

  MMAX : Row dimension of array DG containing Jacobian of constraints. MMAX must be at least one and greater or equal to M.

- int N

  N : Number of optimization variables.

- int NMAX

  NMAX : Row dimension of C. NMAX must be at least two and greater than N.

- int MNN2

  MNN2 : Must be equal to M+N+N+2.

- double ∗ X

  X(NMAX,L) : Initially, the first column of X has to contain starting values for the optimal solution. On return, X is replaced by the current iterate. In the driving program the row dimension of X has to be equal to NMAX. X is used internally to store L different arguments for which function values should be computed simultaneously.

- double ∗ F

  F(L) : On return, F(1) contains the final objective function value. F is used also to store L different objective function values to be computed from L iterates stored in X.

- double ∗ G

  G(MMAX,L) : On return, the first column of G contains the constraint function values at the final iterate X. In the driving program the row dimension of G has to be equal to MMAX. G is used internally to store L different set of constraint function values to be computed from L iterates stored in X.

- double ∗ DF

  DF(NMAX) : DF contains the current gradient of the objective function. In case of numerical differentiation and a distributed system (L>1), it is recommended to apply parallel evaluations of F to compute DF.

- double ∗ DG

  DG(MMAX,NMAX) : DG contains the gradients of the active constraints (ACTIVE(J)=.true.) at a current iterate X. The remaining rows are filled with previously computed gradients. In the driving program the row dimension of DG has to be equal to MMAX.

- double ∗ U

  U(MNN2) : U contains the multipliers with respect to the actual iterate stored in the first column of X. The first M locations contain the multipliers of the M nonlinear constraints, the subsequent N locations the multipliers of the lower bounds, and the final N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.

- double ∗ C

  C(NMAX,NMAX) : On return, C contains the last computed approximation of the Hessian matrix of the Lagrangian function stored in form of an LDL decomposition. C contains the lower triangular factor of an LDL factorization of the final quasi-Newton matrix (without diagonal elements, which are always one). In the driving program, the row dimension of C has to be equal to NMAX.

- double ∗ D

  D(NMAX) : The elements of the diagonal matrix of the LDL decomposition of the quasi-Newton matrix are stored in the one-dimensional array D.

- double ACC

  ACC : The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be smaller than the accuracy by which gradients are computed.

- double ACCQP

  ACCQP : The tolerance is needed for the QP solver to perform several tests, for example whether optimality conditions are satisfied or whether a number is considered as zero or not. If ACCQP is less or equal to zero, then the machine precision is computed by NLPQLP and subsequently multiplied by 1.0D+4.

- double STPMIN

  STPMIN : Minimum steplength in case of L>1. Recommended is any value in the order of the accuracy by which functions are computed. The value is needed to compute a steplength reduction factor by STPMIN∗∗(1/L-1). If STPMIN<=0, then STPMIN=ACC is used.

- int MAXFUN

  MAXFUN : The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20). MAXFUN is only needed in case of L=1, and must not be greater than 50.

- int MAXIT

  MAXIT : Maximum number of outer iterations, where one iteration corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g. 100).

- int MAX_NM

  MAX_NM : Stack size for storing merit function values at previous iterations for non-monotone line search (e.g. 10). In case of MAX_NM=0, monotone line search is performed.

- double TOL_NM

TOL_NM : Relative bound for increase of merit function value, if line search is not successful during the very first step. Must be non-negative (e.g. 0.1).

- int IPRINT

  IPRINT : Specification of the desired output level. IPRINT = 0 : No output of the program. IPRINT = 1 : Only a final convergence analysis is given. IPRINT = 2 : One line of intermediate results is printed in each iteration. IPRINT = 3 : More detailed information is printed in each iteration step, e.g. variable, constraint and multiplier values. IPRINT = 4 : In addition to 'IPRINT=3', merit function and steplength values are displayed during the line search.

- int MODE

  MODE : The parameter specifies the desired version of NLPQLP. MODE = 0 : Normal execution (reverse communication!). MODE = 1 : The user wants to provide an initial guess for the multipliers in U and for the Hessian of the Lagrangian function in C and D in form of an LDL decomposition.

- int IOUT

  IOUT : Integer indicating the desired output unit number, i.e. all write-statements start with 'WRITE(IOUT,... '.

- int IFAIL

  IFAIL : The parameter shows the reason for terminating a solution process. Initially IFAIL must be set to zero. On return IFAIL could contain the following values: IFAIL =-2 : Compute gradient values w.r.t. the variables stored in first column of X, and store them in DF and DG. Only derivatives for active constraints ACTIVE(J)=.TRUE. need to be computed. Then call NLPQLP again, see below. IFAIL =-1 : Compute objective fn and all constraint values subject the variables found in the first L columns of X, and store them in F and G. Then call NLPQLP again, see below. IFAIL = 0 : The optimality conditions are satisfied. IFAIL = 1 : The algorithm has been stopped after MAXIT iterations. IFAIL = 2 : The algorithm computed an uphill search direction. IFAIL = 3 : Underflow occurred when determining a new approxi- mation matrix for the Hessian of the Lagrangian. IFAIL = 4 : The line search could not be terminated successfully. IFAIL = 5 : Length of a working array is too short. More detailed error information is obtained with 'IPRINT>0'. IFAIL = 6 : There are false dimensions, for example M>MMAX, N>=NMAX, or MNN2<>M+N+N+2. IFAIL = 7 : The search direction is close to zero, but the current iterate is still infeasible. IFAIL = 8 : The starting point violates a lower or upper bound. IFAIL = 9 : Wrong input parameter, i.e., MODE, LDL decomposition in D and C (in case of MODE=1), IPRINT, IOUT IFAIL = 10 : Internal inconsistency of the quadratic subproblem, division by zero. IFAIL > 100 : The solution of the quadratic programming subproblem has been terminated with an error message and IFAIL is set to IFQL+100, where IFQL denotes the index of an inconsistent constraint.

- double * WA

  WA(LWA) : WA is a real working array of length LWA.

- int LWA

  LWA : LWA value extracted from NLPQLP20.f.

- int * KWA

  KWA(LKWA) : The user has to provide working space for an integer array.

- int LKWA

  LKWA : LKWA should be at least N+10.

- int * ACTIVE

ACTIVE(LACTIV) : The logical array shows a user the constraints, which NLPQLP considers to be active at the last computed iterate, i.e. G(J,X) is active, if and only if ACTIVE(J)=.TRUE., J=1,...,M.

- int LACTIVE

  LACTIV : The length LACTIV of the logical array should be at least 2∗M+10.

- int LQL

  LQL : If LQL = .TRUE., the quadratic programming subproblem is to be solved with a full positive definite quasi-Newton matrix. Otherwise, a Cholesky decomposition is performed and updated, so that the subproblem matrix contains only an upper triangular factor.

- SizetList nonlinIneqConMappingIndices

  a list of indices for referencing the DAKOTA nonlinear inequality constraints used in computing the corresponding NLPQL constraints.

- RealList nonlinIneqConMappingMultipliers

  a list of multipliers for mapping the DAKOTA nonlinear inequality constraints to the corresponding NLPQL constraints.

- RealList nonlinIneqConMappingOffsets

  a list of offsets for mapping the DAKOTA nonlinear inequality constraints to the corresponding NLPQL constraints.

- SizetList linIneqConMappingIndices

  a list of indices for referencing the DAKOTA linear inequality constraints used in computing the corresponding NLPQL constraints.

- RealList linIneqConMappingMultipliers

  a list of multipliers for mapping the DAKOTA linear inequality constraints to the corresponding NLPQL constraints.

- RealList linIneqConMappingOffsets

  a list of offsets for mapping the DAKOTA linear inequality constraints to the corresponding NLPQL constraints.

10.67.1 Detailed Description

Wrapper class for the NLPQLP optimization library, Version 2.0.

****

AN IMPLEMENTATION OF A SEQUENTIAL QUADRATIC PROGRAMMING METHOD FOR SOLVING NONLINEAR OPTIMIZATION PROBLEMS BY DISTRIBUTED COMPUTING AND NON-MONOTONE LINE SEARCH

This subroutine solves the general nonlinear programming problem

minimize F(X) subject to G(J,X) = 0 , J=1,...,ME G(J,X) >= 0 , J=ME+1,...,M XL <= X <= XU

and is an extension of the code NLPQLD. NLPQLP is specifically tuned to run under distributed systems. A new input parameter L is introduced for the number of parallel computers, that is the number of function calls to be executed simultaneously. In case of L=1, NLPQLP is identical to NLPQLD. Otherwise the line search is modified to allow L parallel function calls in advance. Moreover the user has the opportunity to used distributed function calls for evaluating gradients.

The algorithm is a modification of the method of Wilson, Han, and Powell. In each iteration step, a linearly constrained quadratic programming problem is formulated by approximating the Lagrangian function quadratically and by linearizing the constraints. Subsequently, a one-dimensional line search is performed with respect to an augmented Lagrangian merit function to obtain a new iterate. Also the modified line search algorithm guarantees convergence under the same assumptions as before.

For the new version, a non-monotone line search is implemented which allows to increase the merit function in case of instabilities, for example caused by round-off errors, errors in gradient approximations, etc.

The subroutine contains the option to predetermine initial guesses for the multipliers or the Hessian of the Lagrangian function and is called by reverse communication.

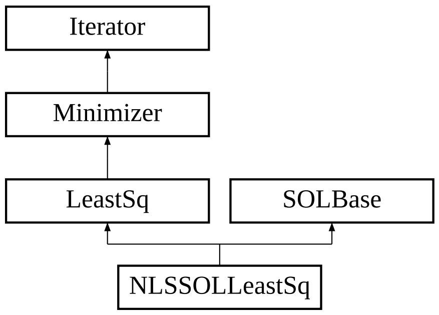
The documentation for this class was generated from the following files:

- NLPQLPOptimizer.H
- NLPQLPOptimizer.C

10.68 NLSSOLLeastSq Class Reference

Wrapper class for the NLSSOL nonlinear least squares library.

Inheritance diagram for NLSSOLLeastSq::



Public Member Functions

- NLSSOLLeastSq (Model &model)

  standard constructor

- ∼NLSSOLLeastSq ()

  destructor

- void minimize_residuals ()

  Used within the least squares branch for minimizing the sum of squares residuals. Redefines the run_iterator virtual function for the least squares branch.

Static Private Member Functions

- static void least_sq_eval (int &mode, int &m, int &n, int &nrowfj, double ∗x, double ∗f, double ∗gradf, int &nstate)

  Evaluator for NLSSOL: computes the values and first derivatives of the least squares terms (passed by function pointer to NLSSOL).

Static Private Attributes

- static NLSSOLLeastSq ∗ nlssolInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

10.68.1 Detailed Description

Wrapper class for the NLSSOL nonlinear least squares library.

The NLSSOLLeastSq class provides a wrapper for NLSSOL, a Fortran 77 sequential quadratic programming library from Stanford University marketed by Stanford Business Associates. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any nonstatic attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows: `max_function_evaluations` is implemented directly in NLSSOLLeastSq's evaluator functions since there is no NLSSOL parameter equivalent, and `max_iterations`, `convergence_tolerance`, `output` verbosity, `verify_level`, `function_precision`, and `linesearch_tolerance` are mapped into NLSSOL's "Major Iteration Limit", "Optimality Tolerance", "Major Print Level" (`verbose:` Major Print Level = 20; `quiet:` Major Print Level = 10), "Verify Level", "Function Precision", and "Linesearch Tolerance" parameters, respectively, using NLSSOL's npoptn() subroutine (as wrapped by npoptn2() from the npoptn_wrapper.f file). Refer to [Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., 1986] for information on NLSSOL's optional input parameters and the npoptn() subroutine.

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

- NLSSOLLeastSq.H
- NLSSOLLeastSq.C

10.69 NoDBBaseConstructor Struct Reference

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

Public Member Functions

- NoDBBaseConstructor (int=0)

  C++ structs can have constructors.

10.69.1 Detailed Description

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

NoDBBaseConstructor is used to overload the constructor used for on-the-fly iterator instantiations in which ProblemDescDB queries cannot be used. Putting this struct here avoids circular dependencies.

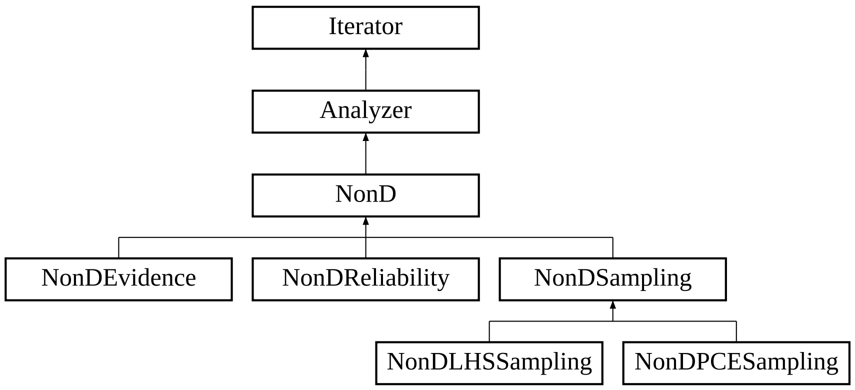
The documentation for this struct was generated from the following file:

- global_defs.h

10.70 NonD Class Reference

Base class for all nondetermistic iterators (the DAKOTA/UQ branch).

Inheritance diagram for NonD::



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 ()

  redefines the main iterator hierarchy virtual function to 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

performs a forward uncertainty propagation of parameter distributions into response statistics

Protected Attributes

- RealMatrix uncertainCorrelations

  uncertain variable correlation matrix (rank correlations for sampling and correlation coefficients for analytic reliability)

- 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 numBetaVars

  number of beta uncertain variables

- size_t numGammaVars

  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

  output probability levels for all response functions resulting from requestedRespLevels

- RealVectorArray computedRelLevels

  output reliability levels for all response functions resulting from requestedRespLevels

- RealVectorArray requestedProbLevels

  requested probability levels for all response functions

- RealVectorArray requestedRelLevels

  requested reliability (beta) levels for all response functions

- RealVectorArray computedRespLevels

  output response levels for all response functions resulting from either requestedProbLevels or requestedRelLevels

- size_t totalLevelRequests

  total number of levels specified within requestedRespLevels, requestedProbLevels, and requestedRelLevels

- bool cdfFlag

  flag for type of probabilities/reliabilities used in mappings: cumulative/CDF (true) or complementary/CCDF (false)

- bool respLevelProbFlag

  flag to indicate mapping of z->p (true) or z->beta (false)

- bool correlationFlag

  flag for indicating if correlation exists among the uncertain variables

- bool strategyFlag

  flag indicating a strategy other than "single_method". Used to compute additional statistics for use at the strategy level or to deactivate additional output not needed for strategy executions.

- Response finalStatistics

final statistics from the uncertainty propagation used in strategies: response means, standard deviations, and probabilities of failure

Private Member Functions

- void distribute_levels (RealVectorArray &levels)

  convenience function for distributing a vector of levels among multiple response functions if a short-hand specification is employed.

10.70.1 Detailed Description

Base class for all nondetermistic iterators (the DAKOTA/UQ branch).

The base class for nondeterministic iterators consolidates uncertain variable data and probabilistic utilities for inherited classes.

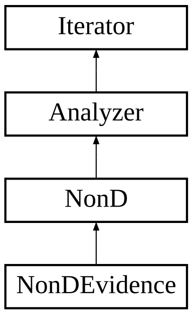
The documentation for this class was generated from the following files:

- DakotaNonD.H
- DakotaNonD.C

10.71 NonDEvidence Class Reference

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

Inheritance diagram for NonDEvidence::



Public Member Functions

- NonDEvidence (Model &model)

  constructor

- ~NonDEvidence ()

  destructor

- void quantify_uncertainty ()

  performs an epistemic uncertainty propagation using Dempster-Shafer evidence theory methods which solve for cumulative distribution functions of belief and plausibility

- void print_results (ostream &s) const

  print the cumulative distribution functions for belief and plausibility

Private Member Functions

- void calculate_basic_prob_intervals ()

  convenience function for encapsulating the calculation of basic probability assignments for input interval combinations

- void calculate_maxmin_per_interval ()

  convenience function for encapsulating the determination of maximum and minimum values within each input interval combination (cell).

- void calculate_cum_belief_plaus ()

  convenience function for determining the cumulative distribution functions of belief and plausbility, based on the max and mins per interval cell

Private Attributes

- 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.

- 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.

10.71.1 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.

10.71.2 Member Data Documentation

10.71.2.1 int NV `[private]`

Size variable for DDS arrays.

NV = number of interval variables

10.71.2.2 int NCMB `[private]`

Size variable for DDS arrays.

NCMB = number of cell combinations

10.71.2.3 int MAXINTVLS `[private]`

Size variable for DDS arrays.

MAXINTVLS = maximum number of intervals per individual interval var

10.71.2.4 Real Y `[private]`

Temporary output variable.

Y = current output to be placed in cell

10.71.2.5 Real∗ BPA [private]

Internal DDS array.

Basic Probability Assignments

10.71.2.6 Real∗ VMIN [private]

Internal DDS array.

Minimum ends of intervals.

10.71.2.7 Real∗ VMAX [private]

Internal DDS array.

Maximum ends of intervals.

10.71.2.8 Real∗ BPAC [private]

Internal DDS array.

Basic Probability Combinations.

10.71.2.9 Real∗ CMIN [private]

Internal DDS Array.

Minimum per cell combination.

10.71.2.10 Real∗ CMAX [private]

Internal DDS Array.

Maximum per cell combination.

10.71.2.11 Real∗ X [private]

Internal DDS Array.

X per cell combination.

10.71.2.12 int∗ NI [private]

Internal DDS array.

Number of intervals per interval variable

10.71.2.13 int* IP [private]

Internal DDS array.

Sort order for combinations

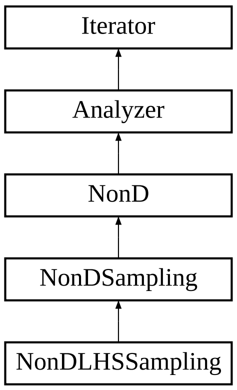
The documentation for this class was generated from the following files:

- NonDEvidence.H
- NonDEvidence.C

10.72 NonDLHSSampling Class Reference

Performs LHS and Monte Carlo sampling for uncertainty quantification.

Inheritance diagram for NonDLHSSampling::



Public Member Functions

- NonDLHSSampling (Model &model)

  constructor

- NonDLHSSampling (Model &model, int samples, int seed)

  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

- void quantify_uncertainty ()

  performs a forward uncertainty propagation by using LHS to generate a set of parameter samples, performing function evaluations on these parameter samples, and computing statistics on the ensemble of results.

- void print_results (ostream &s) const

  print the final statistics

Private Attributes

- bool allVarsFlag

  flags DACE mode using all variables

- bool varBasedDecompFlag

  flags computation of VBD

10.72.1 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.

10.72.2 Constructor & Destructor Documentation

10.72.2.1 NonDLHSSampling (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.

10.72.2.2 NonDLHSSampling (Model & *model*, int *samples*, int *seed*)

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.

10.72.2.3 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).

10.72.3 Member Function Documentation

10.72.3.1 void quantify_uncertainty () `[virtual]`

performs a forward uncertainty propagation by using LHS to generate a set of parameter samples, performing function evaluations on these parameter samples, and computing statistics on the ensemble of results.

Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.

Implements NonD.

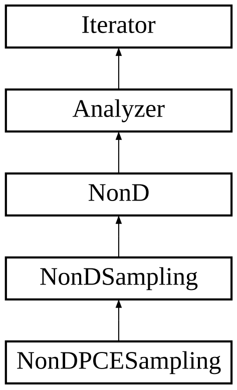
The documentation for this class was generated from the following files:

- NonDLHSSampling.H
- NonDLHSSampling.C

10.73 NonDPCESampling Class Reference

Stochastic finite element approach to uncertainty quantification using polynomial chaos expansions.

Inheritance diagram for NonDPCESampling::



Public Member Functions

- NonDPCESampling (Model &model)

  constructor

- ∼NonDPCESampling ()

  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

- RealVectorArray coeffArray

  Array containing Polynomial Chaos coefficients, one real vector per response function.

- int highestOrder

  Highest order of Hermite Polynomials in Expansion.

- int numChaos

Number of terms in Polynomial Chaos Expansion.

10.73.1 Detailed Description

Stochastic finite element approach to uncertainty quantification using polynomial chaos expansions.

The NonDPCE class uses a polynomial chaos expansion (PCE) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the HermiteSurf and HermiteChaos classes to perform the PCE.

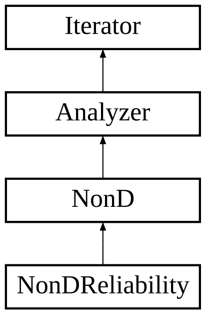
The documentation for this class was generated from the following files:

- NonDPCESampling.H
- NonDPCESampling.C

10.74 NonDReliability Class Reference

Class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDReliability::



Public Member Functions

- NonDReliability (Model &model)

  constructor

- ~NonDReliability ()

  destructor

- void quantify_uncertainty ()

  performs an uncertainty propagation using analytical reliability methods which solve constrained optimization problems to obtain approximations of the cumulative distribution function of response

- void print_results (ostream &s) const

  print the approximate mean, standard deviation, and importance factors when using the mean value method or the CDF/CCDF information when using MPP-search-based reliability methods

- String uses_method () const

  return name of active MPP optimizer

- void method_recourse ()

  perform an MPP optimizer method switch due to a detected conflict

Private Member Functions

- void initialize_random_variables ()

convenience fn for initializing ranVarType, ranVarMeansX, ranVarStdDevsX

- void initial_taylor_series ()

  convenience function for performing the initial limit state Taylor-series approximation

- void mean_value ()

  convenience function for encapsulating the simple Mean Value computation of approximate statistics and importance factors

- void mpp_search ()

  convenience function for encapsulating the reliability methods that employ a search for the most probable point (AMV, AMV+, FORM, SORM)

- void initialize_class_data ()

  convenience function for initializing class scope arrays

- void initialize_level_data ()

  convenience function for initializing/warm starting MPP search data for each response function prior to level 0

- void initialize_mpp_search_data ()

  convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function

- void update_mpp_search_data (const Variables &vars_star, const Response &resp_star)

  convenience function for updating MPP search data for each z/p/beta level for each response function

- void update_level_data (RealVector &final_stats, RealMatrix &final_stat_grads)

  convenience function for updating z/p/beta level data and final statistics following MPP convergence

- void update_limit_state_surrogate ()

  convenience function for passing the latest variables/response data to limitStateSurrogate

- void assign_mean_data ()

  update mostProbPointX/U, computedRespLevel, fnGradX/U, and fnHessX/U from ranVarMeansX/U, fnValsMeanX, fnGradsMeanX, and fnHessiansMeanX

- void g_eval (int &mode, const Epetra_SerialDenseVector &u)

  convenience function for evaluating fnVal(u), fnGradU(u), and fnHessU(u) as required by RIA_constraint_eval() and PMA_objective_eval()

- 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

- void trans_U_to_X (const Epetra_SerialDenseVector &u_vars, Epetra_SerialDenseVector &x_vars)

  Transformation routine from u-space of uncorrelated standard normal variables to x-space of correlated random variables.

- void trans_U_to_Z (const Epetra_SerialDenseVector &u_vars, Epetra_SerialDenseVector &z_vars)

  Transformation routine from u-space of uncorrelated standard normal variables to z-space of correlated standard normal variables.

- void trans_Z_to_X (const Epetra_SerialDenseVector &z_vars, Epetra_SerialDenseVector &x_vars)

  Transformation routine from z-space of correlated standard normal variables to x-space of correlated random variables.

- void trans_X_to_U (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseVector &u_vars)

  Transformation routine from x-space of correlated random variables to u-space of uncorrelated standard normal variables.

- void trans_X_to_Z (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseVector &z_vars)

  Transformation routine from x-space of correlated random variables to z-space of correlated standard normal variables.

- void trans_Z_to_U (Epetra_SerialDenseVector &z_vars, Epetra_SerialDenseVector &u_vars)

  Transformation routine from z-space of correlated standard normal variables to u-space of uncorrelated standard normal variables.

- 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_hess_X_to_U (const Epetra_SerialSymDenseMatrix &fn_hess_x, Epetra_SerialSymDense-Matrix &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 jacobian_dX_dS (const Epetra_SerialDenseVector &x_vars, Epetra_SerialDenseMatrix &jacobian_-xs)

Design Jacobian of x(u,s) mapping obtained from differentiation of trans_U_to_X() 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)

  Computes numerical dx/ds and dz/ds Jacobians as requested by xs and zs booleans.

- void hessian_d2X_dU2 (const Epetra_SerialDenseVector &x_vars, Array< Epetra_SerialSymDenseMatrix > &hessian_xu)

  Hessian of x(u) mapping obtained from dZ/dU^T d^2X/dZ^2 dZ/dU.

- void hessian_d2X_dZ2 (const Epetra_SerialDenseVector &x_vars, Array< Epetra_SerialSymDenseMatrix > &hessian_xz)

  Hessian of x(z) mapping obtained from differentiation of jacobian_dX_dZ().

- void trans_correlations ()

  As part of the Nataf distribution model (Der Kiureghian & Liu, 1986), this procedure modifies the user-specified correlation matrix (corrMatrix) and decomposes it into its Cholesky factor (corrCholeskyFactor).

- 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)

  set a particular random variable distribution parameter and update derived quantities

- Real probability (const Real &beta)

  Convert beta to a probability using either a first-order or second-order integration.

- Real reliability (const Real &p)

  Convert probability to beta using the inverse of a first-order or second-order integration.

- void principal_curvatures ()

  Compute the kappaU vector of principal curvatures from fnHessU.

- Real phi (const Real &beta)

  Standard normal density function.

- Real Phi (const Real &beta)

  Standard normal cumulative distribution function.

- Real Phi_inverse (const Real &p)

Inverse of standard normal cumulative distribution function.

- Real erf_inverse (const Real &p)

  Inverse of error function used in Phi_inverse().

- Real cdf_beta_Pinv (const Real &normcdf, const Real &alpha, const Real &beta)

  Inverse of standard beta CDF (not supported by GSL).

Static Private Member Functions

- static void RIA_objective_eval (int &mode, int &n, double ∗u, double &f, double ∗grad_f, int &)

  static function used by NPSOL as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of (norm u)^2.

- static void RIA_constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int ∗needc, double ∗u, double ∗c, double ∗cjac, int &nstate)

  static function used by NPSOL as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of G(u) = response level.

- static void PMA_objective_eval (int &mode, int &n, double ∗u, double &f, double ∗grad_f, int &)

  static function used by NPSOL as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of G(u).

- static void PMA_constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int ∗needc, double ∗u, double ∗c, double ∗cjac, int &nstate)

  static function used by NPSOL as the constraint function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of (norm u)^2 = beta^2.

- static void RIA_objective_eval (int mode, int n, const NEWMAT::ColumnVector &u, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode)

  static function used by OPT++ as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of (norm u)^2.

- static void RIA_constraint_eval (int mode, int n, const NEWMAT::ColumnVector &u, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, int &result_mode)

  static function used by OPT++ as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of G(u) = response level.

- static void PMA_objective_eval (int mode, int n, const NEWMAT::ColumnVector &u, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode)

static function used by OPT++ as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of G(u).

- static void PMA_constraint_eval (int mode, int n, const NEWMAT::ColumnVector &u, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, int &result_mode)

  static function used by OPT++ as the constraint function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of (norm u)^2 = beta^2.

Private Attributes

- Approximation limitStateSurrogate

  Approximation instance used for TANA-3 and Taylor series limit state approximations.

- size_t numRelAnalyses

  number of invocations of quantify_uncertainty()

- size_t approxIters

  number of AMV+/TANA approximation cycles for the current respFnCount/levelCount

- bool approxConverged

  indicates convergence of approximation-based iterations

- Epetra_SerialDenseVector fnGradX

  actual x-space gradient for current function from most recent response evaluation

- Epetra_SerialDenseVector fnGradU

  u-space gradient for current function updated from fnGradX and Jacobian dx/du

- Epetra_SerialSymDenseMatrix fnHessX

  actual x-space Hessian for current function from most recent response evaluation

- Epetra_SerialSymDenseMatrix fnHessU

  u-space Hessian for current function updated from fnHessX and Jacobian dx/du

- Epetra_SerialDenseVector kappaU

  principal curvatures derived from eigenvalues of orthonormal transformation of fnHessU

- Real modelFnVal

  x-space/u-space value for current function from (linearized) model used in RIA/PMA objective/constraint evaluators

- Epetra_SerialDenseVector modelFnGradU

  u-space gradient for current function from (linearized) model used in RIA/PMA objective/constraint evaluators

- Epetra_SerialSymDenseMatrix modelFnHessU

  u-space Hessian for current function from (linearized) model used in RIA/PMA objective/constraint evaluators

- 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

  response function values evaluated at u=0 (for first-order integration, p=0.5 -> median function values). Used to determine the sign of beta.

- 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

  array of converged MPP's in u-space for level 0. Used for warm-starting initialPtU within RBDO.

- RealMatrix prevFnGradDLev0

  matrix of limit state sensitivities w.r.t. inactive/design variables for level 0. Used for warm-starting initialPtU within RBDO.

- RealMatrix prevFnGradULev0

  matrix of limit state sensitivities w.r.t. active/uncertain variables for level 0. Used for warm-starting initialPtU within RBDO.

- RealVector prevICVars

  previous design vector. Used for warm-starting initialPtU within RBDO.

- IntArray prevCumASVLev0

  accumulation (using |=) of all previous design ASV's from requested finalStatistics. Used to detect availability of prevFnGradDLev0 data for warm-starting initialPtU within RBDO.

- IntArray ranVarType

  vector of indices indicating the type of each uncertain variable

- Epetra_SerialDenseVector ranVarMeansX

vector of means for all uncertain random variables in x-space

- Epetra_SerialDenseVector ranVarMeansU

  vector of means for all uncertain random variables in u-space

- Epetra_SerialDenseVector ranVarStdDevsX

  vector of standard deviations for all uncertain random variables in x-space

- Epetra_SerialSymDenseMatrix corrMatrix

  Epetra copy of uncertainCorrelations.

- Epetra_SerialDenseMatrix corrCholeskyFactor

  cholesky factor of corrMatrix

- 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

- short mppSearchType

  the MPP search type selection: MV, x/u-space AMV, x/u-space AMV+, or FORM

- bool npsolFlag

  flag representing the optimization MPP search algorithm 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

- short integrationOrder

  integration order (1 or 2) provided by `integration` *specification*

- short taylorOrder

  order of Taylor series approximations (1 or 2) in MV/AMV/AMV+ derived from hessianType

- RealMatrix impFactor

  importance factors predicted by MV

- int npsolDerivLevel

  derivative level for NPSOL executions (1 = analytic grads of objective fn, 2 = analytic grads of constraints, 3 = analytic grads of both).

- const Real Pi

  the value for Pi used in several numerical routines

Static Private Attributes

- static NonDReliability $*$ nondRelInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

10.74.1 Detailed Description

Class for the reliability methods within DAKOTA/UQ.

The NonDReliability class implements the following reliability methods through the support of different limit state approximation and integration options: mean value (MV), advanced mean value method (AMV) in x- or u-space, iterated advanced mean value method (AMV+) in x- or u-space, first order reliability method (FORM), and second order reliability method (SORM). The AMV/AMV+/FORM/SORM variants employ an optimizer (currently NPSOL SQP or OPT++ NIP) to solve an equality-constrained optimization problem for the most probable point (MPP).

10.74.2 Member Function Documentation

10.74.2.1 void initialize_random_variables () `[private]`

convenience fn for initializing ranVarType, ranVarMeansX, ranVarStdDevsX

Build ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations.

10.74.2.2 void initial_taylor_series () `[private]`

convenience function for performing the initial limit state Taylor-series approximation

An initial first- or second-order Taylor-series approximation is required for MV/AMV/AMV+/TANA or for the case where meanStats or stdDevStats (from MV) are required within finalStatistics for subIterator usage of NonDReliability.

10.74.2.3 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.

10.74.2.4 void initialize_level_data () `[private]`

convenience function for initializing/warm starting MPP search data for each response function prior to level 0

For a particular response function prior to the first z/p/beta level, initialize/warm-start optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

10.74.2.5 void initialize_mpp_search_data () `[private]`

convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function

For a particular response function at a particular z/p/beta level, warm-start or reset the optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

10.74.2.6 void update_mpp_search_data (const Variables & *vars_star*, const Response & *resp_star*) `[private]`

convenience function for updating MPP search data for each z/p/beta level for each response function

Includes case-specific logic for updating MPP search data for the AMV/AMV+/TANA/NO_APPROX methods.

10.74.2.7 void update_level_data (RealVector & *final_stats*, RealMatrix & *final_stat_grads*) `[private]`

convenience function for updating z/p/beta level data and final statistics following MPP convergence

Updates computedRespLevels/computedProbLevels/computedRelLevels, final_stats/final_stat_grads, warm start, and graphics data.

10.74.2.8 void g_eval (int & *mode*, const Epetra_SerialDenseVector & *u*) `[private]`

convenience function for evaluating fnVal(u), fnGradU(u), and fnHessU(u) as required by RIA_constraint_eval() and PMA_objective_eval()

Convenience function for evaluating the value/gradient/Hessian of G(u). Attributes containing actual data from response evaluations:

- fnValsMeanX, computedRespLevel, fnGradX, fnHessX Attributes used in evaluator fns that may involve approximations:
- modelFnVal, modelFnGradU, modelFnHessU It is important to keep these separate, since fnGradU is used in a number of places (warm start projections, sensitivities, and SORM integrations at converged MPP's) and should not defined from approximations.

10.74.2.9 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.

10.74.2.10 void trans_U_to_X (const Epetra_SerialDenseVector & *u_vars*, Epetra_SerialDenseVector & *x_vars*) `[private]`

Transformation routine from u-space of uncorrelated standard normal 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.

10.74.2.11 void trans_U_to_Z (const Epetra_SerialDenseVector & *u_vars*, Epetra_SerialDenseVector & *z_vars*) `[private]`

Transformation routine from u-space of uncorrelated standard normal 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).

10.74.2.12 void trans_Z_to_X (const Epetra_SerialDenseVector & *z_vars*, Epetra_SerialDenseVector & *x_vars*) `[private]`

Transformation routine from z-space of correlated standard normal 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

10.74.2.13 void trans_X_to_U (const Epetra_SerialDenseVector & *x_vars*, Epetra_SerialDenseVector & *u_vars*) `[private]`

Transformation routine from x-space of correlated random 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.

10.74.2.14 void trans_X_to_Z (const Epetra_SerialDenseVector & *x_vars*, Epetra_SerialDenseVector & *z_vars*) `[private]`

Transformation routine from x-space of correlated random 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.

10.74.2.15 void trans_Z_to_U (Epetra_SerialDenseVector & *z_vars*, Epetra_SerialDenseVector & *u_vars*) `[private]`

Transformation routine from z-space of correlated standard normal 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).

10.74.2.16 void trans_grad_X_to_U (const Epetra_SerialDenseVector & *fn_grad_x*, Epetra_SerialDenseVector & *fn_grad_u*, const Epetra_SerialDenseVector & *x_vars*) `[private]`

Transformation routine for gradient vector from x-space to u-space.

This procedure tranforms 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.

10.74.2.17 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*) `[private]`

Transformation routine for Hessian matrix from x-space to u-space.

This procedure tranforms 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.

10.74.2.18 void jacobian_dX_dU (const Epetra_SerialDenseVector & *x_vars*, Epetra_SerialDenseMatrix & *jacobian_xu*) `[private]`

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.

10.74.2.19 void jacobian_dX_dZ (const Epetra_SerialDenseVector & *x_vars*, Epetra_SerialDenseMatrix & *jacobian_xz*) `[private]`

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.

10.74.2.20 void jacobian_dU_dX (const Epetra_SerialDenseVector & *x_vars*, Epetra_SerialDenseMatrix & *jacobian_ux*) `[private]`

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.

10.74.2.21 void jacobian_dZ_dX (const Epetra_SerialDenseVector & *x_vars*, Epetra_SerialDenseMatrix & *jacobian_zx*) `[private]`

Jacobian of z(x) mapping obtained from differentiation of 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.

10.74.2.22 void jacobian_dX_dS (const Epetra_SerialDenseVector & *x_vars*, Epetra_SerialDenseMatrix & *jacobian_xs*) `[private]`

Design Jacobian of x(u,s) mapping obtained from differentiation of 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.

10.74.2.23 void numerical_design_jacobian (const Epetra_SerialDenseVector & *x_vars*, bool *xs*, Epetra_SerialDenseMatrix & *num_jacobian_xs*, bool *zs*, Epetra_SerialDenseMatrix & *num_jacobian_zs*) `[private]`

Computes numerical dx/ds and dz/ds Jacobians as requested by xs 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).

10.74.2.24 void hessian_d2X_dU2 (const Epetra_SerialDenseVector & *x_vars*, Array< Epetra_SerialSymDenseMatrix > & *hessian_xu*) `[private]`

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.

10.74.2.25 void hessian_d2X_dZ2 (const Epetra_SerialDenseVector & *x_vars*, Array< Epetra_SerialSymDenseMatrix > & *hessian_xz*) `[private]`

Hessian of x(z) mapping obtained from differentiation of 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.

10.74.2.26 void trans_correlations () `[private]`

As part of the Nataf distribution model (Der Kiureghian & Liu, 1986), this procedure modifies the user-specified correlation matrix (corrMatrix) and decomposes it into its Cholesky factor (corrCholeskyFactor).

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
- corrCholeskyFactor: 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 tranformations. All other cases are approximations with some error as noted below.

10.74.2.27 Real probability (const Real & *beta*) `[private]`

Convert beta to a probability using either a first-order or second-order integration.

Converts beta into a probability using either first-order (FORM) or second-order (SORM) integration. The SORM calculation first calculates the principal curvatures at the MPP (using the approach in Ch. 8 of Haldar & Mahadevan), and then applies correction formulations from the literature (Breitung, Hohenbichler/Rackwitz, Tvedt, Hong).

10.74.2.28 Real reliability (const Real & *p*) `[private]`

Convert probability to beta using the inverse of a first-order or second-order integration.

Converts a probability into a reliability using the inverse of the first-order or second-order integrations implemented in NonDReliability::probability(beta).

10.74.2.29 Real Phi (const Real & *beta*) `[inline, private]`

Standard normal cumulative distribution function.

returns a probability < 0.5 for negative beta and a probability > 0.5 for positive beta.

10.74.2.30 Real Phi_inverse (const Real & *p*) `[inline, private]`

Inverse of standard normal cumulative distribution function.

returns a negative beta for probability < 0.5 and a positive beta for probability > 0.5.

10.74.2.31 Real cdf_beta_Pinv (const Real & *normcdf*, const Real & *alpha*, const Real & *beta*) `[private]`

Inverse of standard beta CDF (not supported by GSL).

Solve is performed in scaled space (for the standard beta distribution).

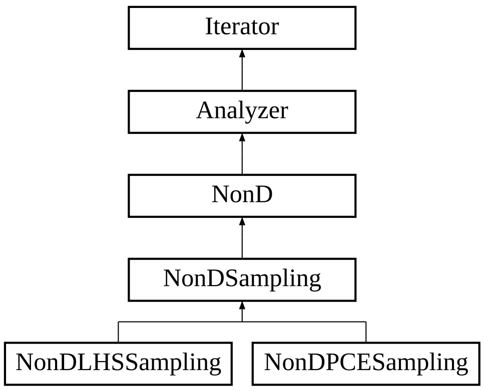
The documentation for this class was generated from the following files:

- NonDReliability.H
- NonDReliability.C

10.75 NonDSampling Class Reference

Base class for common code between NonDLHSSampling and NonDPCESampling.

Inheritance diagram for NonDSampling::



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 (NoDBBaseConstructor, 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 ()

  Uses run_lhs() to generate a set of samples from the distributions in userDefinedModel. In the usual mode, this will be called once. In variance-based decomposition or replicated LHS, it may be called several times.

- void get_parameter_sets (const RealVector &lower_bnds, const RealVector &upper_bnds)

  Uses run_lhs() to generate a set of uniform samples over lower_bnds/upper_bnds.

- void run_lhs (const RealVector &all_l_bnds, const RealVector &all_u_bnds, const RealVector &n_-means, const RealVector &n_std_devs, const RealVector &n_l_bnds, const RealVector &n_u_bnds, const RealVector &ln_means, const RealVector &ln_std_devs, 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 IntVector &num_intervals, const RealVector &interval_probs, const RealVector &interval_bounds)

  generates the desired set of parameter samples from within user-specified probabilistic distributions. Supports both old and new LHS libraries. Used by NonDLHSSampling and NonDPCESampling.

- void compute_statistics (const RealVectorArray &samples)

  computes mean, standard deviation, and probability of failure for the samples input

- void compute_correlations (const RealVectorArray &all_c_vars, const RealVectorArray &all_fns)

  computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank

- void simple_corr (Epetra_SerialDenseMatrix &total_data, const int &num_obs, const int &num_corr, const bool &rank_on)

  computes simple correlations

- void partial_corr (Epetra_SerialDenseMatrix &total_data, const int &num_obs, const int &num_corr, const bool &rank_on)

  computes partial correlations

- void print_statistics (ostream &s) const

  prints the mean, standard deviation, and probability of failure statistics computed in compute_statistics()

Static Protected Member Functions

- static bool rank_sort (const int &x, const int &y)

  sort algorithm to compute ranks for rank correlations

Protected Attributes

- int samplesSpec

  user specification of number of samples

- int numSamples

  the number of samples to evaluate

- String sampleType

  the sample type: "lhs" or "random"

- bool statsFlag

  flags computation/output of statistics

- bool allDataFlag

  flags update of allVariables/allResponses

- size_t numActiveVars

  total number of variables published to LHS

- size_t numDesignVars

  number of design variables (treated as uniform distribution within design variable bounds for DACE usage of NonDSampling)

- size_t numStateVars

  number of state variables (treated as uniform distribution within state variable bounds for DACE usage of NonDSampling)

- bool varyPattern

  flag for generating a sequence of seed values within multiple run_lhs() calls so that the run_lhs() executions (e.g., for surrogate-based optimization) are repeatable but not correlated.

Private Member Functions

- void check_error (const int &err_code, const char ∗err_source) const

  checks the return codes from LHS routines and aborts if an error is returned

Private Attributes

- const int originalSeed

  the user seed specification (default is 0)

- int randomSeed

  the current random number seed

- size_t numLHSRuns

  counter for number of executions of run_lhs() for this object

- RealVector mean95CIDeltas

  Plus/minus deltas on response function means for 95% confidence intervals (calculated in compute_statistics()).

- RealVector stdDev95CILowerBnds

  Lower bound for 95% confidence interval on std deviation (calculated in compute_statistics()).

- RealVector stdDev95CIUpperBnds

  Upper bound for 95% confidence interval on std deviation (calculated in compute_statistics()).

- Epetra_SerialDenseMatrix simpleCorr

  matrix to hold simple raw correlations

- Epetra_SerialDenseMatrix simpleRankCorr

  matrix to hold simple rank correlations

- Epetra_SerialDenseMatrix partialCorr

  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.

10.75.1 Detailed Description

Base class for common code between NonDLHSSampling and NonDPCESampling.

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.

10.75.2 Constructor & Destructor Documentation

10.75.2.1 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.

10.75.2.2 NonDSampling (NoDBBaseConstructor, Model & *model*, int *samples*, int *seed*) [protected]

alternate constructor for sample generation and evaluation "on the fly"

This alternate constructor is used by NonDEvidence for generation and evaluation of on-the-fly sample sets.

10.75.2.3 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.

10.75.3 Member Function Documentation

10.75.3.1 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 ApproximationInterface::build_global_approximation() 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.

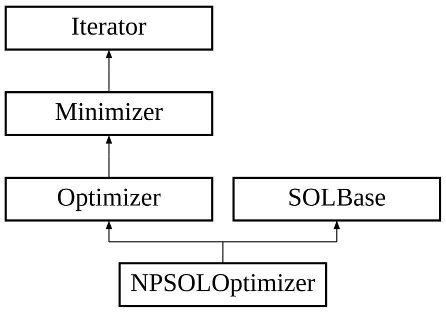
The documentation for this class was generated from the following files:

- NonDSampling.H
- NonDSampling.C

10.76 NPSOLOptimizer Class Reference

Wrapper class for the NPSOL optimization library.

Inheritance diagram for NPSOLOptimizer::



Public Member Functions

- NPSOLOptimizer (Model &model)

  standard constructor

- NPSOLOptimizer (const RealVector &initial_point, const RealVector &var_lower_bnds, const RealVector &var_upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_lower_bnds, const RealVector &lin_ineq_upper_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_-targets, const RealVector &nonlin_ineq_lower_bnds, const RealVector &nonlin_ineq_upper_bnds, const RealVector &nonlin_eq_targets, void(∗user_obj_eval)(int &, int &, double ∗, double &, double ∗, int &), void(∗user_con_eval)(int &, int &, int &, int &, int ∗, double ∗, double ∗, double ∗, int &), const int &derivative_level, const Real &conv_tol)

  alternate constructor for instantiations "on the fly"

- ∼NPSOLOptimizer ()

  destructor

- void find_optimum ()

  Used within the optimizer branch for computing the optimal solution. Redefines the run_iterator virtual function for the optimizer branch.

Private Member Functions

- void find_optimum_on_model ()

  called by find_optimum for setUpType == "model"

- void find_optimum_on_user_functions ()

  called by find_optimum for setUpType == "user_functions"

Static Private Member Functions

- static void objective_eval (int &mode, int &n, double ∗x, double &f, double ∗gradf, int &nstate)

  OBJFUN in NPSOL manual: computes the value and first derivatives of the objective function (passed by function pointer to NPSOL).

Private Attributes

- String setUpType

  controls iteration mode: "model" (normal usage) or "user_functions" (user-supplied functions mode for "on the fly" instantiations). NonDReliability currently uses the user_functions mode.

- RealVector initialPoint

  holds initial point passed in for "user_functions" mode.

- RealVector lowerBounds

  holds variable lower bounds passed in for "user_functions" mode.

- RealVector upperBounds

  holds variable upper bounds passed in for "user_functions" mode.

- void(∗ userObjectiveEval)(int &, int &, double ∗, double &, double ∗, int &)

  holds function pointer for objective function evaluator passed in for "user_functions" mode.

- void(∗ userConstraintEval)(int &, int &, int &, int &, int ∗, double ∗, double ∗, double ∗, int &)

  holds function pointer for constraint function evaluator passed in for "user_functions" mode.

Static Private Attributes

- static NPSOLOptimizer ∗ npsolInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

10.76.1 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.

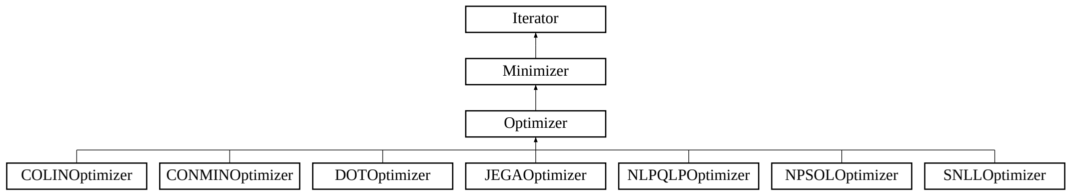
The documentation for this class was generated from the following files:

- NPSOLOptimizer.H
- NPSOLOptimizer.C

10.77 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy.

Inheritance diagram for Optimizer::



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, size_t num_cv, 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)

  set the relative weightings for multiple objective functions. Used by ConcurrentStrategy for Pareto set optimization.

- void derived_initialize_scaling ()

  provides derived class-specific portions of scaling initialization since Optimizer and LeastSq iterators have obj fn. and residual scales, respectively

- virtual void find_optimum ()=0

  Used within the optimizer branch for computing the optimal solution. Redefines the run_iterator virtual function for the optimizer branch.

- Response multi_objective_modify (const Response &raw_response) const

  forward mapping: maps multiple objective functions to a single objective for single-objective optimizers

- const RealVector & multi_objective_retrieve (const Variables &vars, const Response &response) const

  inverse mapping: retrieves values for multiple objective functions from the solution of a single-objective optimizer

Protected Attributes

- size_t numObjectiveFunctions

  number of objective functions

- RealVector multiObjWeights

  user-specified weights for multiple objective functions

Friends

- class COLINApplication

  a COLINOptimizer uses a COLINApplication object to perform the function evaluations

10.77.1 Detailed Description

Base class for the optimizer branch of the iterator hierarchy.

The Optimizer class provides common data and functionality for DOTOptimizer, NPSOLOptimizer, SNLLOptimizer, and COLINOptimizer.

10.77.2 Constructor & Destructor Documentation

10.77.2.1 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.

10.77.3 Member Function Documentation

10.77.3.1 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.

10.77.3.2 void print_results (ostream & *s*) const `[protected, virtual]`

Redefines default iterator results printing to include optimization results (objective functions and constraints).

Reimplemented from Iterator.

10.77.3.3 Response multi_objective_modify (const Response & *raw_response*) const `[protected]`

forward mapping: maps multiple objective functions to a single 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.

10.77.3.4 const RealVector & multi_objective_retrieve (const Variables & *vars*, const Response & *response*) const `[protected]`

inverse mapping: retrieves values for multiple objective functions 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

10.78 ParallelConfiguration Class Reference

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

Public Member Functions

- ParallelConfiguration ()

  default constructor

- ParallelConfiguration (const ParallelConfiguration &pl)

  copy constructor

- ∼ParallelConfiguration ()

  destructor

- ParallelConfiguration & operator= (const ParallelConfiguration &pl)

  assignment operator

- const ParallelLevel & w_parallel_level () const

  return the ParallelLevel corresponding to wPLIter

- const ParallelLevel & si_parallel_level () const

  return the ParallelLevel corresponding to siPLIter

- const ParallelLevel & ie_parallel_level () const

  return the ParallelLevel corresponding to iePLIter

- const ParallelLevel & ea_parallel_level () const

  return the ParallelLevel corresponding to eaPLIter

Private Member Functions

- void assign (const ParallelConfiguration &pl)

  assign the attributes of the incoming pl to this object

Private Attributes

- short numParallelLevels

  number of parallel levels

- ParLevLIter wPLIter

  list iterator for MPI_COMM_WORLD (not strictly required, but improves modularity by avoiding explicit usage of MPI_COMM_WORLD)

- ParLevLIter siPLIter

  list iterator for concurrent iterator partitions (there may be more than one per parallel configuration instance)

- ParLevLIter iePLIter

  list iterator identifying the iterator-evaluation parallelLevel (there can only be one)

- ParLevLIter eaPLIter

  list iterator identifying the evaluation-analysis parallelLevel (there can only be one)

Friends

- class ParallelLibrary

  the ParallelLibrary class has special access priveleges in order to streamline implementation

10.78.1 Detailed Description

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

Rather than containing the multilevel parallel configuration directly, ParallelConfiguration instead provides a set of list iterators which point into a combined list of ParallelLevels. This approach allows different configurations to reuse ParallelLevels without copying them. A list of ParallelConfigurations is contained in ParallelLibrary (ParallelLibrary::parallelConfigurations).

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

- ParallelLibrary.H

10.79 ParallelLevel Class Reference

Container class for the data associated with a single level of communicator partitioning.

Public Member Functions

- ParallelLevel ()
  default constructor
- ParallelLevel (const ParallelLevel &pl)
  copy constructor
- ∼ParallelLevel ()
  destructor
- ParallelLevel & operator= (const ParallelLevel &pl)
  assignment operator
- bool dedicated_master_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

  the ParallelLibrary class has special access priveleges in order to streamline implementation

10.79.1 Detailed Description

Container class for the data associated with a single level of communicator partitioning.

A list of these levels is contained in ParallelLibrary (ParallelLibrary::parallelLevels), which defines all of the parallelism levels across one or more multilevel parallelism configurations.

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

- ParallelLibrary.H

10.80 ParallelLibrary Class Reference

Class for partitioning multiple levels of parallelism and managing message passing within these levels.

Public Member Functions

- ParallelLibrary (int &argc, char $**$&argv)

  stand-alone mode constructor

- ParallelLibrary ()

  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)

  specify output streams and restart file(s) using command line 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)

  specify output streams and restart file(s) using external 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_e (int &data)

  broadcast an integer across an evaluation communicator
- void bcast_a (int &data)

  broadcast an integer across an analysis communicator
- void bcast_si (int &data)

  broadcast an integer across a strategy-iterator intra communicator
- void bcast_w (MPIPackBuffer &send_buff)

  broadcast a packed buffer across MPI_COMM_WORLD
- void bcast_i (MPIPackBuffer &send_buff)

  broadcast a packed buffer across an iterator communicator
- void bcast_e (MPIPackBuffer &send_buff)

  broadcast a packed buffer across an evaluation communicator
- void bcast_a (MPIPackBuffer &send_buff)

  broadcast a packed buffer across an analysis communicator
- void bcast_si (MPIPackBuffer &send_buff)

  broadcast a packed buffer across a strategy-iterator intra communicator
- void bcast_w (MPIUnpackBuffer &recv_buff)

  matching receive for packed buffer broadcast across MPI_COMM_WORLD
- void bcast_i (MPIUnpackBuffer &recv_buff)

  matching receive for packed buffer bcast across an iterator communicator
- void bcast_e (MPIUnpackBuffer &recv_buff)

  matching receive for packed buffer bcast across an evaluation communicator
- void bcast_a (MPIUnpackBuffer &recv_buff)

  matching receive for packed buffer bcast across an analysis communicator

- void bcast_si (MPIUnpackBuffer &recv_buff)

  matching recv for packed buffer bcast across a strat-iterator intra comm

- void barrier_w ()

  enforce MPI_Barrier on MPI_COMM_WORLD

- void barrier_i ()

  enforce MPI_Barrier on an iterator communicator

- void barrier_e ()

  enforce MPI_Barrier on an evaluation communicator

- void barrier_a ()

  enforce MPI_Barrier on an analysis communicator

- void reduce_sum_ea (double ∗local_vals, double ∗sum_vals, const int &num_vals)

  compute a sum over an eval-analysis intra-communicator using MPI_Reduce

- void reduce_sum_a (double ∗local_vals, double ∗sum_vals, const int &num_vals)

  compute a sum over an analysis communicator using MPI_Reduce

- void test (MPI_Request &request, int &test_flag, MPI_Status &status)

  test a nonblocking send/receive request for completion

- void wait (MPI_Request &request, MPI_Status &status)

  wait for a nonblocking send/receive request to complete

- void waitall (const int &num_recvs, MPI_Request ∗&recv_reqs)

  wait for all messages from a series of nonblocking receives

- void waitsome (const int &num_sends, MPI_Request ∗&recv_requests, int &num_recvs, int ∗&index_-array, MPI_Status ∗&status_array)

  wait for at least one message from a series of nonblocking receives but complete all that are available

- void free (MPI_Request &request)

  free an MPI_Request

- const int & world_size () const

  return worldSize

- const int & world_rank () const

  return worldRank

- bool mpirun_flag () const

  return mpirunFlag

- bool is_null () const

  return dummyFlag

- Real parallel_time () const

  returns current MPI wall clock time

- void parallel_configuration_iterator (const ParConfigLIter &pc_iter)

  set the current ParallelConfiguration node

- const ParConfigLIter & parallel_configuration_iterator () const

  return the current ParallelConfiguration node

- const ParallelConfiguration & parallel_configuration () const

  return the current ParallelConfiguration instance

- size_t num_parallel_configurations () const

  returns the number of entries in parallelConfigurations

- bool parallel_configuration_is_complete ()

  identifies if the current ParallelConfiguration has been fully populated

- void increment_parallel_configuration ()

  add a new node to parallelConfigurations and increment currPCIter

- bool w_parallel_level_defined () const

  test current parallel configuration for definition of world parallel level

- bool si_parallel_level_defined () const

  test current parallel configuration for definition of strategy-iterator parallel level

- bool ie_parallel_level_defined () const

  test current parallel configuration for definition of iterator-evaluation parallel level

- bool ea_parallel_level_defined () const

  test current parallel configuration for definition of evaluation-analysis parallel level

- Array< MPI_Comm > analysis_intra_communicators ()

  return the set of analysis intra communicators for all parallel configurations (used for setting up direct simulation interfaces prior to execution time).

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)

  split a parent communicator into a dedicated master processor and num_servers child communicators

- bool split_communicator_peer_partition (const ParallelLevel &parent_pl, ParallelLevel &child_pl, const int &proc_remainder)

  split a parent communicator into num_servers peer child communicators (no dedicated master processor)

- bool resolve_inputs (int &num_servers, int &procs_per_server, const int &avail_procs, int &proc_-remainder, const int &max_concurrency, const int &capacity_multiplier, const String &default_config, const String &scheduling_override)

  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 (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

  the complete set of parallelism levels for managing multilevel parallelism among one or more configurations

- List< ParallelConfiguration > parallelConfigurations

  the set of parallel configurations which manage list iterators for indexing into parallelLevels

- ParLevLIter currPLIter

  list iterator identifying the current node in parallelLevels

- ParConfigLIter currPCIter

  list iterator identifying the current node in parallelConfigurations

10.80.1 Detailed Description

Class for partitioning multiple levels of parallelism and managing message passing within these levels.

The ParallelLibrary class encapsulates all of the details of performing message passing within multiple levels of parallelism. It provides functions for partitioning of levels according to user configuration input and functions for passing messages within and across MPI communicators for each of the parallelism levels. If support for other message-passing libraries beyond MPI becomes needed (PVM, ...), then ParallelLibrary would be promoted to a base class with virtual functions to encapsulate the library-specific syntax.

10.80.2 Constructor & Destructor Documentation

10.80.2.1 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.

10.80.2.2 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.

10.80.2.3 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.

10.80.3 Member Function Documentation

10.80.3.1 void specify_outputs_restart (CommandLineHandler & *cmd_line_handler*)

specify output streams and restart file(s) using command line 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.

10.80.3.2 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.

10.80.3.3 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.

10.80.3.4 void increment_parallel_configuration () `[inline]`

add a new node to parallelConfigurations and increment currPCIter

Called from the ParallelLibrary ctor and from Model::init_communicators(). An increment is performed for each Model initialization except the first (which inherits the world and strategy-iterator parallel levels from the first partial configuration).

10.80.3.5 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.

10.80.3.6 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*) `[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

10.81 ParamResponsePair Class Reference

Container class for a variables object, a response object, and an evaluation id.

Public Member Functions

- ParamResponsePair ()

  default constructor

- ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, bool deep_copy=false)

  alternate constructor for temporaries

- ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, const int eval_id, bool deep_copy=true)

  standard constructor for history uses

- ParamResponsePair (const ParamResponsePair &pair)

  copy constructor

- ∼ParamResponsePair ()

  destructor

- ParamResponsePair & operator= (const ParamResponsePair &pair)

  assignment operator

- void read (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)

read a ParamResponsePair object from the binary restart stream

- void write (BoStream &s) const

    write a ParamResponsePair object to the binary restart stream

- void read (MPIUnpackBuffer &s)

    read a ParamResponsePair object from a packed MPI buffer

- void write (MPIPackBuffer &s) const

    write a ParamResponsePair object to a packed MPI buffer

- int eval_id () const

    return the evaluation identifier

- const 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

    the interface used to generate the response object. Used in ParamResponsePair::vars_set_compare to prevent duplicate detection on results from different interfaces.

- int evalId

    the function evaluation identifier (assigned from ApplicationInterface::fnEvalId)

Friends

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

  equality operator

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

  inequality operator

10.81.1 Detailed Description

Container class for a variables object, a response object, and an evaluation id.

ParamResponsePair provides a container class for association of the input for a particular function evaluation (a variables object) with the output from this function evaluation (a response object), along with an evaluation identifier. This container defines the basic unit used in the data_pairs 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.

10.81.2 Constructor & Destructor Documentation

10.81.2.1 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).

10.81.2.2 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.

10.81.3 Member Function Documentation

10.81.3.1 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.

10.81.3.2 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.

10.81.4 Member Data Documentation

10.81.4.1 String idInterface `[private]`

the interface used to generate the response object. Used in ParamResponsePair::vars_set_compare to prevent duplicate 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().

10.81.4.2 int evalId `[private]`

the function evaluation identifier (assigned from 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().

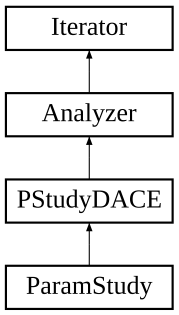
The documentation for this class was generated from the following files:

- ParamResponsePair.H
- ParamResponsePair.C

10.82 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 ()

  computes stepVector and numSteps from initialPoint, finalPoint, and either numSteps or stepLength (pStudyType is 1 or 2)

- void vector_loop (const RealVector &start, const RealVector &step_vect, const int &num_steps)

  performs the parameter study by looping from start in num_steps increments of step_vect. Total number of evaluations is num_steps + 1.

- void sample (const RealVector &list_of_points)

  performs the parameter study by sampling from a list of points

- void centered_loop (const RealVector &start, const Real &percent_delta, const int &deltas_per_variable)

performs a number of plus and minus offsets for each parameter centered about start

- void multidim_loop (const IntArray &var_partitions)

  performs vector_loops recursively in multiple dimensions

- void recurse (int nloop, int nindex, IntArray &current_index, const IntArray &max_index, const RealVector &start, const RealVector &step_vect)

  used by multidim_loop to enable a variable number of nested loops

Private Attributes

- RealVector listOfPoints

  list 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

- int pStudyType

  internal code for parameter study type: -1 (list), 1,2,3 (different vector specifications), 4 (centered), or 5 (multidim)

- int deltasPerVariable

  number of offsets in the plus and the minus direction for each variable in a centered_parameter_study

- bool nestedFlag

  flag set by parameter studies which call other parameter studies in loops

- Real stepLength

  the Cartesian length of multidimensional steps in vector_parameter_study (a specification option)

- Real percentDelta

  size of relative offsets in percent for each variable in a centered_parameter_study

- IntArray variablePartitions

  number of partitions for each variable in a multidim_parameter_study

- int psCounter

  class-scope counter (needed for asynchronous multidim_loop)

10.82.1 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.

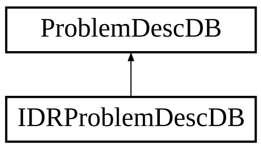
The documentation for this class was generated from the following files:

- ParamStudy.H
- ParamStudy.C

10.83 ProblemDescDB Class Reference

The database containing information parsed from the DAKOTA input file.

Inheritance diagram for ProblemDescDB::



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)

  parses the input file and populates the problem description database. This version reads from the dakota input filename passed with the "-input" option on the DAKOTA command line.

- void manage_inputs (const char ∗dakota_input_file)

  parses the input file and populates the problem description database. This version reads from the dakota input filename passed in.

- void check_input ()

  verifies that there was at least one of each of the required keywords in the dakota input file. Used by manage_inputs().

- void set_db_list_nodes (const String &method_tag)

set dataMethodIter based on a method identifier string to activate a particular method specification in dataMethod-List and use pointers from this method specification to set all other list iterators.

- void set_db_list_nodes (const size_t &method_index)

  set dataMethodIter based on an index within dataMethodList to activate a particular method specification and use pointers from this method specification to set all other list iterators.

- void set_db_method_node (const size_t &method_index)

  set dataMethodIter based on an index within dataMethodList to activate a particular method specification (only).

- size_t get_db_method_node ()

  return the index of the active node in dataMethodList

- void set_db_model_nodes (const String &model_tag)

  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on the model identifier string

- void set_db_model_nodes (const size_t &model_index)

  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on an index within dataModelList

- size_t get_db_model_node ()

  return the index of the active node in dataModelList

- void set_db_variables_node (const String &variables_tag)

  set dataVariablesIter based on the variables identifier string

- void set_db_interface_node (const String &interface_tag)

  set dataInterfaceIter based on the interface identifier string

- void set_db_responses_node (const String &responses_tag)

  set dataResponsesIter based on the responses identifier string

- ParallelLibrary & parallel_library () const

  return the parallelLib reference

- IteratorList & iterator_list ()

  return a list of all Iterator objects that have been instantiated

- ModelList & model_list ()

  return a list of all Model objects that have been instantiated

- VariablesList & variables_list ()

  return a list of all Variables objects that have been instantiated

- InterfaceList & interface_list ()

  return a list of all Interface objects that have been instantiated

- ResponseList & response_list ()

  return a list of all Response objects that have been instantiated

- const 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 & get_drv (const String &entry_name) const

  get a RealVector out of the database based on an identifier string

- const IntVector & get_div (const String &entry_name) const

  get a IntVector out of the database based on an identifier string

- const IntArray & get_dia (const String &entry_name) const

  get a IntArray out of the database based on an identifier string

- const RealMatrix & get_drm (const String &entry_name) const

  get a RealMatrix out of the database based on an identifier string

- const RealVectorArray & get_drva (const String &entry_name) const

  get a RealVectorArray out of the database based on an identifier string

- const IntList & get_dil (const String &entry_name) const

  get a IntList out of the database based on an identifier string

- const StringArray & get_dsa (const String &entry_name) const

  get a StringArray out of the database based on an identifier string

- const String2DArray & get_ds2a (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

Protected Member Functions

- ProblemDescDB (BaseConstructor, ParallelLibrary &parallel_lib)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- virtual void derived_manage_inputs (const char ∗dakota_input_file)

  parses the input file and populates the problem description database. This version reads from the dakota input filename passed in.

Protected Attributes

- DataStrategy strategySpec

  the strategy specification (only one allowed) resulting from a call to strategy_kwhandler() or insert_node()

- List< DataMethod > dataMethodList

  list of method specifications, one for each call to method_kwhandler() or insert_node()

- List< DataModel > dataModelList

  list of model specifications, one for each call to model_kwhandler() or insert_node()

- List< DataVariables > dataVariablesList

  list of variables specifications, one for each call to variables_kwhandler() or insert_node()

- List< DataInterface > dataInterfaceList

  list of interface specifications, one for each call to interface_kwhandler() or insert_node()

- List< DataResponses > dataResponsesList

  list of responses specifications, one for each call to responses_kwhandler() or insert_node()

- size_t strategyCntr

  counter for strategy specifications used in check_input

Private Member Functions

- ProblemDescDB ∗ get_db (ParallelLibrary &parallel_lib)

  Used by the standard envelope constructor to instantiate the correct letter class.

- void send_db_buffer ()

  MPI send of a large buffer containing strategySpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage_inputs().

- void receive_db_buffer ()

  MPI receive of a large buffer containing strategySpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage_inputs().

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

  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

  prevents use of get_<type> data retrieval functions prior to a set_db_list_nodes invocation

- ProblemDescDB ∗ dbRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing dbRep

10.83.1 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.

10.83.2 Constructor & Destructor Documentation

10.83.2.1 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.

10.83.2.2 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.

10.83.2.3 ProblemDescDB (const ProblemDescDB & *db*)

copy constructor

Copy constructor manages sharing of dbRep and incrementing of referenceCount.

10.83.2.4 ~ProblemDescDB ()

destructor

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

10.83.2.5 ProblemDescDB (BaseConstructor, ParallelLibrary & *parallel_lib*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_db() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_db() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~ProblemDescDB).

10.83.3 Member Function Documentation

10.83.3.1 **ProblemDescDB operator= (const ProblemDescDB & *db*)**

assignment operator

Assignment operator decrements referenceCount for old dbRep, assigns new dbRep, and increments reference-Count for new dbRep.

10.83.3.2 **void manage_inputs (CommandLineHandler & *cmd_line_handler*)**

parses the input file and populates the problem description database. This version reads from the dakota input filename passed with the "-input" option on the DAKOTA command line.

Manage command line inputs using the CommandLineHandler class and parse the input file.

10.83.3.3 **void manage_inputs (const char ∗ *dakota_input_file*)**

parses the input file and populates the problem description database. This version reads from the dakota input filename passed in.

Parse the input file.

10.83.3.4 **ProblemDescDB ∗ get_db (ParallelLibrary & *parallel_lib*)** `[private]`

Used by the standard envelope constructor to instantiate the correct letter class.

Initializes dbRep to the appropriate derived type. The standard derived class constructors are invoked.

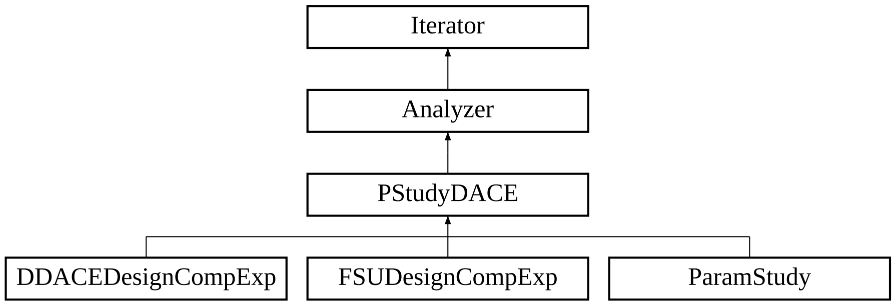
The documentation for this class was generated from the following files:

- ProblemDescDB.H
- ProblemDescDB.C

10.84 PStudyDACE Class Reference

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

Inheritance diagram for PStudyDACE::



Protected Member Functions

- PStudyDACE (Model &model)

    constructor

- ∼PStudyDACE ()

    destructor

- void run ()

    run the iterator; portion of run_iterator()

- const Variables & variable_results () const

    return the final iterator solution (variables)

- const Response & response_results () const

    return the 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

  best constraint violations found during the study. In the current approach, constraint violation reduction takes strict 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

10.84.1 Detailed Description

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

The PStudyDACE base class manages common data and functions, such as those involving the best solutions located during the parameter set evaluations or the printing of final results.

10.84.2 Member Function Documentation

10.84.2.1 void run () `[inline, protected, 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.

10.84.2.2 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

10.85 Response Class Reference

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

Public Member Functions

- Response ()

  default constructor

- Response (const Variables &vars, const ProblemDescDB &problem_db)

  standard constructor built from problem description database

- Response (const ActiveSet &set)

  alternate constructor using limited data

- Response (const Response &response)

  copy constructor

- ∼Response ()

  destructor

- Response operator= (const Response &response)

  assignment operator

- size_t num_functions () const

  return the number of response functions

- const ActiveSet & active_set () const

  return the active set

- void active_set (const ActiveSet &set)

  set the active set

- const IntArray & active_set_request_vector () const

  return the active set request vector

- void active_set_request_vector (const IntArray &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 & fn_tags () const

    return the function identifier strings

- void fn_tags (const StringArray &tags)

    set the function identifier strings

- const RealVector & function_values () const

    return the function values

- void function_values (const RealVector &function_vals)

    set the function values

- const RealMatrix & function_gradients () const

    return the function gradients

- void function_gradients (const RealMatrix &function_grads)

    set the function gradients

- const RealMatrixArray & function_hessians () const

    return the function Hessians

- void function_hessians (const RealMatrixArray &function_hessians)

    set the 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)

  Used in place of operator= when only results data updates are desired (functionValues/functionGradients/functionHessians are updated, ASV/tags/id's/etc. are not). Care is taken to allow 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)

  Overloaded form which allows update from components of a response object. Care is taken to allow different derivative array sizing.

- 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

10.85.1 Detailed Description

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

The Response class is a container class for an abstract set of functions (functionValues) and their first (functionGradients) and second (functionHessians) derivatives. The functions may involve objective and constraint functions (optimization data set), least squares terms (parameter estimation data set), or generic response functions (uncertainty quantification data set). It is not currently part of a class hierarchy, since the abstraction has been sufficiently general and has not required specialization. For memory efficiency, it employs the "handle-body idiom" approach to reference counting and representation sharing (see Coplien "Advanced C++", p. 58), for which Response serves as the handle and ResponseRep serves as the body.

10.85.2 Constructor & Destructor Documentation

10.85.2.1 Response ()

default constructor

Need a populated problem description database to build a meaningful Response object, so set the responseRep=NULL in default constructor for efficiency. This then requires a check on NULL in the copy constructor, assignment operator, and destructor.

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

- DakotaResponse.H
- DakotaResponse.C

10.86 ResponseRep Class Reference

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

Private Member Functions

- ResponseRep ()

  default constructor

- ResponseRep (const Variables &vars, const ProblemDescDB &problem_db)

  standard constructor built from problem description database

- ResponseRep (const ActiveSet &set)

  alternate constructor using limited data

- ∼ResponseRep ()

  destructor

- void read (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 ()

  return the number of doubles active in response. Used for sizing double∗ response_data arrays passed into read_-data and write_data.

- void read_data (double ∗response_data)

  read from an incoming double∗ array

- void write_data (double ∗response_data)

  write to an incoming double∗ array

- void overlay (const Response &response)

  add incoming response to functionValues/Gradients/Hessians

- void 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_flag, bool hess_flag)

  rehapes response data arrays

- void reset ()

  resets all response data to zero

- void reset_inactive ()

  resets all inactive response data to zero

- void active_set_request_vector (const IntArray &asrv)

  set the active set request vector and verify consistent number of response functions

- void active_set_derivative_vector (const IntArray &asdv)

  set the active set derivative vector and reshape functionGradients/functionHessians if needed

Private Attributes

- int referenceCount

  number of handle objects sharing responseRep

- RealVector functionValues

  abstract set of functions

- RealMatrix functionGradients

  first derivatives

- RealMatrixArray functionHessians

  second derivatives

- ActiveSet responseActiveSet

  copy of the ActiveSet used by the Model to generate a Response instance

- StringArray fnTags

  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

10.86.1 Detailed Description

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

The ResponseRep class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (Response) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (ResponseRep) actually contains the response data (functionValues, functionGradients, 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).

10.86.2 Constructor & Destructor Documentation

10.86.2.1 ResponseRep (const Variables & *vars*, const ProblemDescDB & *problem_db*) `[private]`

standard constructor built from problem description database

The standard constructor used by Dakota::ModelRep.

10.86.2.2 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). fnTags 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 bestResponses and bestResponses needs fnTags for I/O, so construction of fnTags has been added.

10.86.3 Member Function Documentation

10.86.3.1 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.).

10.86.3.2 void write (ostream & *s*) const `[private]`

write a responseRep object to an ostream

ASCII version of write.

10.86.3.3 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.

10.86.3.4 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.

10.86.3.5 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.

10.86.3.6 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.

10.86.3.7 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.

10.86.3.8 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.

10.86.3.9 void read (MPIUnpackBuffer & *s*) `[private]`

read a responseRep object from a packed MPI buffer

UnpackBuffer version differs from BiStream version in the omission of fnTags. Master processor retains function tags and interface ids and communicates asv and response data only with slaves.

10.86.3.10 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 fnTags. The master processor retains tags and ids and communicates asv and response data only with slaves.

10.86.3.11 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, fnTags, etc. Also, care is taken to account for differences in derivative variable matrix sizing.

10.86.3.12 void reshape (const size_t & *num_fns*, const size_t & *num_params*, bool *grad_flag*, bool *hess_flag*) `[private]`

rehapes response data arrays

Reshape functionValues, functionGradients, and functionHessians according to num_fns, num_params, grad_flag, and hess_flag.

10.86.3.13 void reset () `[private]`

resets all response data to zero

Reset all numerical response data (not tags, ids, or active set) to zero.

10.86.3.14 void reset_inactive () `[private]`

resets all inactive response data to zero

Used to clear out any inactive data left over from previous evaluations.

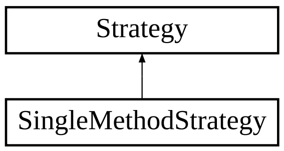
The documentation for this class was generated from the following files:

- DakotaResponse.H
- DakotaResponse.C

10.87 SingleMethodStrategy Class Reference

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

Inheritance diagram for SingleMethodStrategy::



Public Member Functions

- SingleMethodStrategy (ProblemDescDB &problem_db)

  constructor

- ∼SingleMethodStrategy ()

  destructor

- void run_strategy ()

  Perform the strategy by executing selectedIterator on userDefinedModel.

- const Variables & variable_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

10.87.1 Detailed Description

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

This strategy executes a single iterator on a single model. Since it does not provide coordination for multiple iterators and models, it can considered to be a "fall-through" strategy in that it allows control to fall through immediately to the iterator.

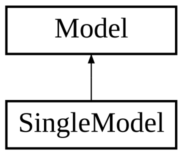
The documentation for this class was generated from the following files:

- SingleMethodStrategy.H
- SingleMethodStrategy.C

10.88 SingleModel Class Reference

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

Inheritance diagram for SingleModel::



Public Member Functions

- SingleModel (ProblemDescDB &problem_db)

  constructor

- ~SingleModel ()

  destructor

Protected Member Functions

- Interface & interface ()

  return userDefinedInterface

- void derived_compute_response (const ActiveSet &set)

  portion of compute_response() specific to SingleModel (invokes a synchronous map() on userDefinedInterface)

- void derived_asynch_compute_response (const ActiveSet &set)

  portion of asynch_compute_response() specific to SingleModel (invokes an asynchronous map() on userDefined-Interface)

- const ResponseArray & derived_synchronize ()

  portion of synchronize() specific to SingleModel (invokes synch() on userDefinedInterface)

- const IntResponseMap & derived_synchronize_nowait ()

  portion of synchronize_nowait() specific to SingleModel (invokes synch_nowait() on userDefinedInterface)

- void component_parallel_mode (int mode)

  SingleModel only supports parallelism in userDefinedInterface, so this virtual function redefinition is simply a sanity check.

- String local_eval_synchronization ()

  return userDefinedInterface synchronization setting

- int local_eval_concurrency ()

  return userDefinedInterface asynchronous evaluation concurrency

- bool derived_master_overload () const

  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to userDefined-Interface)

- void derived_init_communicators (const int &max_iterator_concurrency)

  set up SingleModel for parallel operations (request forwarded to userDefinedInterface)

- void derived_init_serial ()

  set up SingleModel for serial operations (request forwarded to userDefinedInterface).

- void reset_communicators ()

  reset communicator partition data for the SingleModel (request forwarded to userDefinedInterface)

- void derived_free_communicators (const int &max_iterator_concurrency)

  deallocate communicator partitions for the SingleModel (request forwarded to userDefinedInterface)

- void serve ()

  Service userDefinedInterface job requests received from the master. Completes when a termination message is received from stop_servers().

- void stop_servers ()

  executed by the master to terminate userDefinedInterface server operations when SingleModel iteration is complete.

- int evaluation_id () const

  return the current evaluation id for the SingleModel (request forwarded to userDefinedInterface)

- void set_evaluation_reference ()

  set the evaluation counter reference points for the SingleModel (request forwarded to userDefinedInterface)

- void print_evaluation_summary (ostream &s, bool minimal_header=false, bool relative_count=true) const

  print the evaluation summary for the SingleModel (request forwarded to userDefinedInterface)

Private Attributes

- Interface userDefinedInterface

  the interface used for mapping variables to responses

10.88.1 Detailed Description

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

The SingleModel class is the simplest of the derived model classes. It provides the capabilities the old 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.

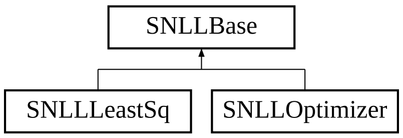
The documentation for this class was generated from the following files:

- SingleModel.H
- SingleModel.C

10.89 SNLLBase Class Reference

Base class for OPT++ optimization and least squares methods.

Inheritance diagram for SNLLBase::



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)

  convenience function for copying local_fn_vals to g; used by constraint evaluator functions

- void copy_con_vals (const NEWMAT::ColumnVector &g, RealVector &local_fn_vals, const size_t &offset)

  convenience function for copying g to local_fn_vals; used in final solution logging

- void copy_con_grad (const RealMatrix &local_fn_grads, NEWMAT::Matrix &grad_g, const size_t &offset)

  convenience function for copying local_fn_grads to grad_g; used by constraint evaluator functions

- void copy_con_hess (const RealMatrixArray &local_fn_hessians, OPTPP::OptppArray< NEWMAT::SymmetricMatrix > &hess_g, const size_t &offset)

  convenience function for copying local_fn_hessians to hess_g; used by constraint evaluator functions

- void snll_pre_instantiate (const String &merit_fn, bool bound_constr_flag, const int &num_constr)

convenience function for setting OPT++ options prior to the method instantiation

- void snll_post_instantiate (const int &num_cv, bool vendor_num_grad_flag, const String &finite_diff_-type, const Real &fdss, const int &max_iter, const int &max_fn_evals, const Real &conv_tol, const Real &grad_tol, const Real &max_step, bool bound_constr_flag, const int &num_constr, bool debug_-output, OPTPP::OptimizeClass ∗the_optimizer, OPTPP::NLP0 ∗nlf_objective, OPTPP::FDNLF1 ∗fd_nlf1, OPTPP::FDNLF1 ∗fd_nlf1_con)

  convenience function for setting OPT++ options after the method instantiation

- void snll_pre_run (OPTPP::NLP0 ∗nlf_objective, OPTPP::NLP ∗nlp_constraint, const RealVector &init_-pt, bool bound_constr_flag, const RealVector &lower_bnds, const RealVector &upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_targets)

  convenience function for OPT++ configuration prior to the method invocation

- void snll_post_run (OPTPP::NLP0 ∗nlf_objective)

  convenience function for setting OPT++ options after the method instantiations

Static Protected Member Functions

- static void init_fn (int n, NEWMAT::ColumnVector &x)

  An initialization mechanism provided by OPT++ (not currently used).

Protected Attributes

- String searchMethod

  value_based_line_search, gradient_based_line_search, trust_region, or tr_pds

- OPTPP::SearchStrategy searchStrat

  enum: LineSearch, TrustRegion, or TrustPDS

- OPTPP::MeritFcn meritFn

  enum: NormFmu, ArgaezTapia, or VanShanno

- bool constantASVFlag

  flags a user selection of active_set_vector == constant. By mapping this into mode override, reliance on duplicate detection can be avoided.

Static Protected Attributes

- static Minimizer ∗ optLSqInstance

pointer to the active base class object instance used within the static evaluator functions in order to avoid the need for static data

- static bool modeOverrideFlag

  flags OPT++ mode override (for combining value, gradient, and Hessian requests)

- static EvalType lastFnEvalLocn

  an enum used to track whether an nlf evaluator or a constraint evaluator was the last location of a function evaluation

- static int lastEvalMode

  copy of mode from constraint evaluators

- static RealVector lastEvalVars

  copy of variables from constraint evaluators

10.89.1 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.

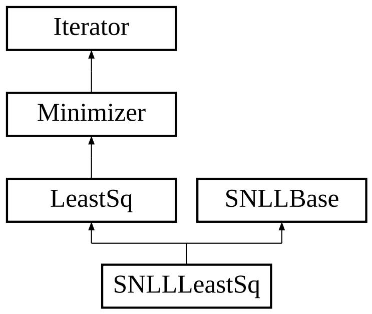
The documentation for this class was generated from the following files:

- SNLLBase.H
- SNLLBase.C

10.90 SNLLLeastSq Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for 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)

objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation.

- static void constraint1_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, int &result_mode)

  constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.

- static void constraint2_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, OPTPP::OptppArray< NEWMAT::SymmetricMatrix > &hess_g, int &result_mode)

  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.

Private Attributes

- 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

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

10.90.1 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/VendorOptimizers/opt++ directory for information on OPT++ class member functions.

10.90.2 Member Function Documentation

10.90.2.1 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]`

objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation.

This nlf2 evaluator function is used for the Gauss-Newton method in order to exploit the special structure of the nonlinear least squares problem. Here, fx = 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).

10.90.2.2 void constraint1_evaluator_gn (int *mode*, int *n*, const NEWMAT::ColumnVector & *x*, NEWMAT::ColumnVector & *g*, NEWMAT::Matrix & *grad_g*, int & *result_mode*) `[static, private]`

constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.

While it does not employ the Gauss-Newton approximation, it is distinct from constraint1_evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with diaggregated Hessian NIPS and is currently active.

10.90.2.3 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]`

constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.

While it does not employ the Gauss-Newton approximation, it is distinct from constraint2_evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with full Newton NIPS and is currently inactive.

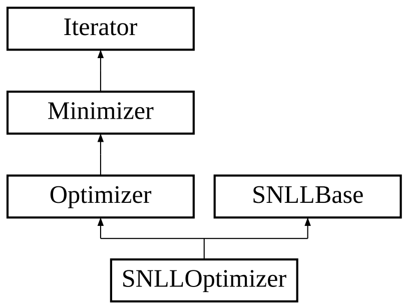
The documentation for this class was generated from the following files:

- SNLLLeastSq.H
- SNLLLeastSq.C

10.91 SNLLOptimizer Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLOptimizer::



Public Member Functions

- SNLLOptimizer (Model &model)

  standard constructor

- 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 &g, NEW-MAT::Matrix &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

- virtual void derived_post_run ()

  invokes SNLLBase::snll_post_run() and performs other solution processing

Static Private Member Functions

- static void nlf0_evaluator (int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, int &result_-mode)

  objective function evaluator function for OPT++ methods which require only function values.

- static void nlf1_evaluator (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode)

  objective function evaluator function which provides function values and gradients to OPT++ methods.

- static void nlf2_evaluator (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, NEWMAT::SymmetricMatrix &hess_f, int &result_mode)

  objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

- static void constraint0_evaluator (int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, int &result_mode)

  constraint evaluator function for OPT++ methods which require 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)

  constraint evaluator function which provides constraint 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)

  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.

Private Attributes

- OPTPP::NLP0 ∗ nlfObjective

  objective NLF base class pointer

- OPTPP::NLP0 ∗ nlfConstraint

  constraint NLF base class pointer

- OPTPP::NLP ∗ nlpConstraint

  constraint NLP pointer

- OPTPP::NLF0 ∗ nlf0

  pointer to objective NLF for nongradient optimizers

- OPTPP::NLF1 ∗ nlf1

  pointer to objective NLF for (analytic) gradient-based optimizers

- OPTPP::NLF1 ∗ nlf1Con

  pointer to constraint NLF for (analytic) gradient-based optimizers

- OPTPP::FDNLF1 ∗ fdnlf1

  pointer to objective NLF for (finite diff) gradient-based optimizers

- OPTPP::FDNLF1 ∗ fdnlf1Con

  pointer to constraint NLF for (finite diff) gradient-based optimizers

- OPTPP::NLF2 ∗ nlf2

  pointer to objective NLF for full Newton optimizers

- OPTPP::NLF2 ∗ nlf2Con

  pointer to constraint NLF for full Newton optimizers

- OPTPP::OptimizeClass ∗ theOptimizer

  optimizer base class pointer

- OPTPP::OptPDS ∗ optpds

  PDS optimizer pointer.

- OPTPP::OptCG ∗ optcg

  CG optimizer pointer.

- OPTPP::OptLBFGS ∗ optlbfgs

  L-BFGS optimizer pointer.

- OPTPP::OptNewton ∗ optnewton

  Newton optimizer pointer.

- OPTPP::OptQNewton ∗ optqnewton

  Quasi-Newton optimizer pointer.

- OPTPP::OptFDNewton ∗ optfdnewton

  Finite Difference Newton opt pointer.

- OPTPP::OptBCNewton ∗ optbcnewton

  Bound constrained Newton opt pointer.

- OPTPP::OptBCQNewton ∗ optbcqnewton

  Bnd constrained Quasi-Newton opt ptr.

- OPTPP::OptBCFDNewton ∗ optbcfdnewton

    Bnd constrained FD-Newton opt ptr.

- OPTPP::OptNIPS ∗ optnips

    NIPS optimizer pointer.

- OPTPP::OptQNIPS ∗ optqnips

    Quasi-Newton NIPS optimizer pointer.

- OPTPP::OptFDNIPS ∗ optfdnips

    Finite Difference NIPS opt pointer.

- String setUpType

    flag for iteration mode: "model" (normal usage) or "user_functions" (user-supplied functions mode for "on the fly" instantiations). NonDReliability currently uses the user_functions mode.

- RealVector initialPoint

    holds initial point passed in for "user_functions" mode.

- RealVector lowerBounds

    holds variable lower bounds passed in for "user_functions" mode.

- RealVector upperBounds

    holds variable upper bounds passed in for "user_functions" mode.

Static Private Attributes

- static SNLLOptimizer ∗ snllOptInstance

    pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

10.91.1 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/VendorOptimizers/opt++ directory for information on OPT++ class member functions.

10.91.2 Constructor & Destructor Documentation

10.91.2.1 SNLLOptimizer (Model & *model*)

standard constructor

This constructor is used for normal instantiations using data from the ProblemDescDB.

10.91.2.2 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(∗)(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::ColumnVector &g, 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.

10.91.3 Member Function Documentation

10.91.3.1 void nlf0_evaluator (int *n*, const NEWMAT::ColumnVector & *x*, NEWMAT::Real & *f*, int & *result_mode*) `[static, private]`

objective function evaluator function for OPT++ methods which require only function values.

For use when DAKOTA computes f and gradients are not directly available. This is used by nongradient-based optimizers such as PDS and by gradient-based optimizers in vendor numerical gradient mode (opt++'s internal finite difference routine is used).

10.91.3.2 void nlf1_evaluator (int *mode*, int *n*, const NEWMAT::ColumnVector & *x*, NEWMAT::Real & *f*, NEWMAT::ColumnVector & *grad_f*, int & *result_mode*) `[static, private]`

objective function evaluator function which provides function values and gradients to OPT++ methods.

For use when DAKOTA computes f and df/dX (regardless of gradientType). Vendor numerical gradient case is handled by nlf0_evaluator.

10.91.3.3 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]`

objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

For use when DAKOTA receives f, df/dX, & d^2f/dx^2 from the ApplicationInterface (analytic only). Finite differencing does not make sense for a full Newton approach, since lack of analytic gradients & Hessian should dictate the use of quasi-newton or fd-newton. Thus, there is no fdnlf2_evaluator for use with full Newton approaches, since it is preferable to use quasi-newton or fd-newton with nlf1. Gauss-Newton does not fit this model; it uses nlf2_evaluator_gn instead of nlf2_evaluator.

10.91.3.4 void constraint0_evaluator (int *n*, const NEWMAT::ColumnVector & *x*, NEWMAT::ColumnVector & *g*, int & *result_mode*) `[static, private]`

constraint evaluator function for OPT++ methods which require only constraint values.

For use when DAKOTA computes g and gradients are not directly available. This is used by nongradient-based optimizers and by gradient-based optimizers in vendor numerical gradient mode (opt++'s internal finite difference routine is used).

10.91.3.5 void constraint1_evaluator (int *mode*, int *n*, const NEWMAT::ColumnVector & *x*, NEWMAT::ColumnVector & *g*, NEWMAT::Matrix & *grad_g*, int & *result_mode*) `[static, private]`

constraint evaluator function which provides constraint values and gradients to OPT++ methods.

For use when DAKOTA computes g and dg/dX (regardless of gradientType). Vendor numerical gradient case is handled by constraint0_evaluator.

10.91.3.6 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*) `[static, private]`

constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.

For use when DAKOTA computes g, dg/dX, & d^2g/dx^2 (analytic only).

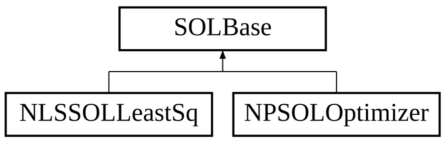
The documentation for this class was generated from the following files:

- SNLLOptimizer.H
- SNLLOptimizer.C

10.92 SOLBase Class Reference

Base class for Stanford SOL software.

Inheritance diagram for SOLBase::



Public Member Functions

- SOLBase ()

  default constructor

- SOLBase (Model &model)

  standard constructor

- ∼SOLBase ()

  destructor

Protected Member Functions

- void allocate_arrays (const int &num_cv, const size_t &num_nln_con, const RealMatrix &lin_ineq_coeffs, const RealMatrix &lin_eq_coeffs)

  Allocates miscellaneous arrays for the SOL algorithms.

- void deallocate_arrays ()

  Deallocates memory previously allocated by allocate_arrays().

- void allocate_workspace (const int &num_cv, const int &num_nln_con, const int &num_lin_con, const int &num_lsq)

  Allocates real and integer workspaces for the SOL algorithms.

- void set_options (bool speculative_flag, bool vendor_num_grad_flag, 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)

  CONFUN in NPSOL manual: computes the values and first derivatives of the nonlinear constraint functions.

Protected Attributes

- int realWorkSpaceSize

  size of realWorkSpace

- int intWorkSpaceSize

  size of intWorkSpace

- RealArray realWorkSpace

  real work space for NPSOL/NLSSOL

- IntArray intWorkSpace

  int work space for NPSOL/NLSSOL

- int nlnConstraintArraySize

  used for non-zero array sizing (nonlinear constraints)

- int linConstraintArraySize

  used for non-zero array sizing (linear constraints)

- RealArray cLambda

  CLAMBDA from NPSOL manual: Langrange multipliers.

- IntArray constraintState

  ISTATE from NPSOL manual: constraint status.

- int informResult

  INFORM from NPSOL manual: optimization status on exit.

- int numberIterations

  ITER from NPSOL manual: number of (major) iterations performed.

- int boundsArraySize

  length of augmented bounds arrays (variable bounds plus linear and nonlinear constraint bounds)

- double ∗ linConstraintMatrixF77

  [A] matrix from NPSOL manual: linear constraint coefficients

- double ∗ upperFactorHessianF77

  [R] matrix from NPSOL manual: upper Cholesky factor of the Hessian of the Lagrangian.

- double ∗ constraintJacMatrixF77

  [CJAC] matrix from NPSOL manual: nonlinear constraint Jacobian

- int fnEvalCntr

  counter for testing against maxFunctionEvals

- size_t constrOffset

  used in constraint_eval() to bridge NLSSOLLeastSq::numLeastSqTerms and NPSOLOptimizer::numObjectiveFunctions

Static Protected Attributes

- static SOLBase ∗ solInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

- static Minimizer ∗ optLSqInstance

  pointer to the active base class object instance used within the static evaluator functions in order to avoid the need for static data

10.92.1 Detailed Description

Base class for Stanford SOL software.

The SOLBase class provides a common base class for NPSOLOptimizer and NLSSOLLeastSq, both of which are Fortran 77 sequential quadratic programming algorithms from Stanford University marketed by Stanford Business Associates.

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

- SOLBase.H
- SOLBase.C

10.93 SortCompare Class Template Reference

Public Member Functions

- SortCompare (bool(∗func)(const T &, const T &))

  Constructor that defines the pointer to function.

- bool operator() (const T &p1, const T &p2) const

  The operator() must be defined. Calls the defined sort_fn.

Private Attributes

- bool(∗ sort_fn)(const T &, const T &)

  Pointer to test function.

10.93.1 Detailed Description

template$<$class T$>$ class Dakota::SortCompare$<$ T $>$

Internal functor used in the sort algorithm to sort using a specified compare method. The class holds a pointer to the sort function.

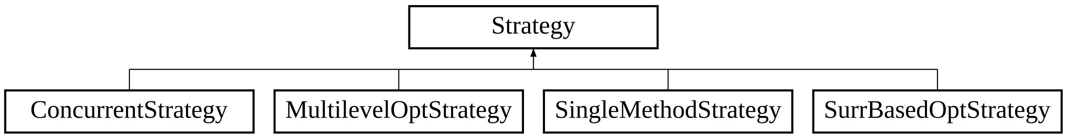
The documentation for this class was generated from the following file:

- DakotaList.H

10.94 Strategy Class Reference

Base class for the strategy class hierarchy.

Inheritance diagram for Strategy::



Public Member Functions

- Strategy ()

  default constructor

- Strategy (ProblemDescDB &problem_db)

  envelope constructor

- Strategy (const Strategy &strat)

  copy constructor

- virtual ~Strategy ()

  destructor

- Strategy operator= (const Strategy &strat)

  assignment operator

- virtual void run_strategy ()

  the run function for the strategy: invoke the iterator(s) on the model(s). Called from main.C.

- virtual const Variables & variable_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)

  Convenience function for invoking an iterator and managing parallelism. This version omits communicator repartitioning. Function must be public due to use by MINLPNode.

- ProblemDescDB & prob_desc_db () const

returns the problem description database (probDescDB)

- ParallelLibrary & parallel_library () const

  returns the parallel library (parallelLib)

Protected Member Functions

- Strategy (BaseConstructor, ProblemDescDB &problem_db)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- 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)

  convenience function for initialization of 2D graphics and 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

  type of strategy: single_method, multi_level, surrogate_based_opt, 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

10.94.1 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.

10.94.2 Constructor & Destructor Documentation

10.94.2.1 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.

10.94.2.2 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.

10.94.2.3 Strategy (const Strategy & *strat*)

copy constructor

Copy constructor manages sharing of strategyRep and incrementing of referenceCount.

10.94.2.4 ∼Strategy () `[virtual]`

destructor

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

10.94.2.5 Strategy (BaseConstructor, ProblemDescDB & *problem_db*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all inherited strategies. get_strategy() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_strategy() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ∼Strategy).

10.94.3 Member Function Documentation

10.94.3.1 Strategy operator= (const Strategy & *strat*)

assignment operator

Assignment operator decrements referenceCount for old strategyRep, assigns new strategyRep, and increments referenceCount for new strategyRep.

10.94.3.2 void run_iterator (Iterator & *the_iterator*, Model & *the_model*)

Convenience function for invoking an iterator and managing parallelism. This version omits communicator repartitioning. Function must be public due to use by MINLPNode.

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.

10.94.3.3 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.

10.94.3.4 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.

10.94.3.5 void initialize_graphics (const Model & *model*) `[protected]`

convenience function for initialization of 2D graphics and 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.

10.94.3.6 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

10.95 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.

10.95.1 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

10.95.2 Member Function Documentation

10.95.2.1 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.

10.95.2.2 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.

10.95.2.3 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.

10.95.2.4 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.

10.95.2.5 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.

10.95.2.6 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.

10.95.2.7 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.

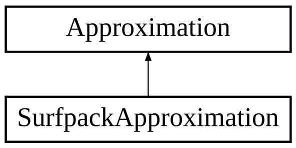
The documentation for this class was generated from the following files:

- DakotaString.H
- DakotaString.C

10.96 SurfpackApproximation Class Reference

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

Inheritance diagram for SurfpackApproximation::



Public Member Functions

- SurfpackApproximation ()

  default constructor

- SurfpackApproximation (ProblemDescDB &problem_db, const size_t &num_acv)

  and are passed as is to Surfpack methods and functions– which expect structures of type std::vector. This is legal as long as the types Surfpack expects really are superclasses of their Dakota counterparts. It will fail if a Real is #defined in Dakota to be a float, because then it will be a subclass of std::vector<float> being passed in as a std::vector<double>. The possible solutions, I think, are: 1. Have the same typedef feature in Surfpack so that everything can be configured to use floats or doubles 2. Explicitly cast each vector component as it passes over the Dakota/Surfpack boundary

- ∼SurfpackApproximation ()

  destructor

Protected Member Functions

- int num_coefficients () const
- void find_coefficients ()

  SurfData object will be created from Dakota's SurrogateDataPoints, and the appropriate Surfpack build method will be invoked.

- const RealVector & approximation_coefficients ()

  Vector of values representing the identity of the Surfpack surface. Return value of this function is not yet meaningful. The format of such a vector of values is not yet defined for all Surfpack classes.

- 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 ()

  If anchor_point is present, create equality constraints from a particular point, gradient, and/or hessian.

- SurfData ∗ surrogates_to_surf_data ()

  copy from SurrogateDataPoint to SurfPoint/SurfData

Private Attributes

- RealVector coefficients

  Vector representation of the Approximation (e.g., polynomial coefficients for linear regression or trained neural network weights). The format of such a vector has not been defined for all Surfpack classes.

- Surface ∗ surface

  The native Surfpack approximation.

- SurfData ∗ surfData

  The data used to build the approximation, in Surfpack format.

- short polyOrder

  order (1, 2, or 3) of a polynomial regression surrogate

10.96.1 Detailed Description

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

The SurfpackApproximation class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SurfpackApproximation builds a Surfpack Surface object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial neural networks, radial basis function network, or multivariate adaptaive regression splines (MARS).

10.96.2 Constructor & Destructor Documentation

10.96.2.1 SurfpackApproximation (ProblemDescDB & *problem_db*, const size_t & *num_acv*)

and are passed as is to Surfpack methods and functions– which expect structures of type std::vector. This is legal as long as the types Surfpack expects really are superclasses of their Dakota counterparts. It will fail if a Real is #defined in Dakota to be a float, because then it will be a subclass of std::vector<float> being passed in as a std::vector<double>. The possible solutions, I think, are: 1. Have the same typedef feature in Surfpack so that everything can be configured to use floats or doubles 2. Explicitly cast each vector component as it passes over 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

10.96.3 Member Function Documentation

10.96.3.1 int num_coefficients () const `[protected, virtual]`

Todo

: Check to make sure that the number of points required does not

Todo

: The reported number of points required is computed in a rather

exceed the bounds for a signed integer ad hoc manner. Do something smarter.

Reimplemented from Approximation.

10.96.3.2 void find_coefficients () `[protected, virtual]`

SurfData object will be created from Dakota's SurrogateDataPoints, and the appropriate Surfpack build method will be invoked.

Todo

Right now, we're completely deleting the old data and then

surfData will be deleted in dtor 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.

10.96.3.3 const RealVector & approximation_coefficients () `[protected, virtual]`

Vector of values representing the identity of the Surfpack surface. Return value of this function is not yet meaningful. The format of such a vector of values is not yet defined for all Surfpack classes.

The value returned from this function is currently meaningless.

Todo

: Provide an appropriate list of coefficients for each surface type

Reimplemented from Approximation.

10.96.3.4 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.

10.96.3.5 void checkForEqualityConstraints () `[private]`

If anchor_point is present, create equality constraints from a particular 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

10.96.3.6 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.

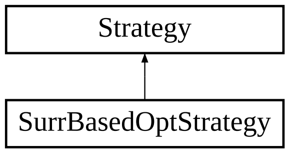
The documentation for this class was generated from the following files:

- SurfpackApproximation.H
- SurfpackApproximation.C

10.97 SurrBasedOptStrategy Class Reference

Strategy for provably-convergent surrogate-based optimization.

Inheritance diagram for SurrBasedOptStrategy::



Public Member Functions

- SurrBasedOptStrategy (ProblemDescDB &problem_db)

  constructor

- ∼SurrBasedOptStrategy ()

  destructor

- void run_strategy ()

  Performs the surrogate-based optimization strategy by optimizing local, global, or hierarchical surrogates over a series of trust regions.

- const Variables & variable_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 (RealVector &c_vars_center, const StringArray &c_vars_labels, 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 (Response &response_center_truth, const RealVector &c_vars_center, const Iterator &dace_iterator, Model &truth_model)

retrieve response_center_truth if possible, evaluate it if not

- void find_center_approx (Response &response_center_approx, const Response &response_center_truth)

  retrieve response_center_approx if possible, evaluate it if not

- void hard_convergence_check (const Response &response_truth, const RealVector &c_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)

  check for hard convergence (norm of projected gradient of merit function near zero)

- void tr_ratio_check (const RealVector &c_vars_center, const RealVector &c_vars_star, const RealVector &tr_lower_bounds, const RealVector &tr_upper_bounds, const Response &response_center_truth, const Response &response_center_approx, const Response &response_star_truth, const Response &response_-star_approx)

  compute trust region ratio for accepting/rejecting SBO iterate and sizing 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)

  compute a Lagrangian function from a set of function values

- void lagrangian_gradient (const RealMatrix &fn_grads, RealVector &lag_grad)

  compute the gradient of the Lagrangian function

- Real augmented_lagrangian_merit (const RealVector &fn_vals)

  compute an augmented Lagrangian function from a set of function values

- void augmented_lagrangian_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, RealVector &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, RealVector &pen_grad)

  compute the gradient of the penalty function

- Real objective (const RealVector &fn_vals)

  compute a single objective value from one or more objective functions

- Real constraint_violation (const RealVector &fn_vals, const Real &constraint_tol)

  compute the constraint violation from a set of function values

- void relax_constraints (const Response &response_truth, const RealVector &c_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)

  relax constraints by updating bounds when current iterate is infeasible

Static Private Member Functions

- static void hom_objective_eval (int &mode, int &n, double ∗u, double &f, double ∗grad_f, int &)

  static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.

- static void hom_constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int ∗needc, double ∗u, double ∗c, double ∗cjac, int &nstate)

  static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation.

Private Attributes

- Model surrogateModel

  the surrogate model (a SurrogateModel object)

- Iterator selectedIterator

  the optimizer used on surrogateModel

- Real trustRegionFactor

  the trust region factor is used to compute the total size of the trust region – it is a percentage, e.g. for trustRegionFactor = 0.1, the actual size of the trust region will be 10% of the global bounds (upper bound - lower bound for each design variable).

- Real minTrustRegionFactor

  a soft convergence control: stop SBO when the trust region factor is reduced below the value of minTrustRegionFactor

- Real convergenceTol

  the optimizer convergence tolerance; used in several SBO hard and soft convergence checks

- Real constraintTol

  a tolerance specifying the distance from a constraint boundary that is allowed before an active constraint is considered to 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)

- bool trConstraintRelax

  flag to use trust region constraint relaxation for infeasible starting points

- int trConstraintRelaxMethod

  type of trust region constraint relaxation for infeasible starting points: NONE (default=0) or HOMOTOPY (1)

- int meritFnType

  type of merit function used in trust region ratio logic: BASIC_PENALTY, ADAPTIVE_PENALTY, BASIC_-LAGRANGIAN, or AUGMENTED_LAGRANGIAN

- int acceptLogic

  type of iterate acceptance test logic: FILTER or TR_RATIO

- RealVectorList sboFilter

  Set of response function vectors defining a filter (objective vs. constraint violation) for iterate selection/rejection.

- RealVector lagrangeMult

  Lagrange multipliers for basic Lagrangian calculations.

- RealVector augLagrangeMult

  Lagrange multipliers for augmented Lagrangian calculations.

- Real penaltyParameter

  the penalization factor for violated constraints used in quadratic penalty calculations; increased in update_penalty()

- int penaltyIterOffset

  iteration offset used to update the scaling of the penalty parameter for adaptive_penalty merit functions

- 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

  number of consecutive candidate point rejections. If the count reaches softConvLimit, stop SBO.

- short softConvLimit

  the limit on consecutive candidate point rejections. If exceeded by softConvCount, stop SBO.

- bool gradientFlag

  flags the use of gradients within the SBO process

- bool hessianFlag

  flags the use of Hessians within the SBO process

- bool correctionFlag

  flags the use of surrogate correction techniques at the center of each trust region

- bool globalApproxFlag

  flags the use of a global data fit surrogate (rsm, ann, mars, kriging)

- bool localApproxFlag

  flags the use of a local data fit surrogate (Taylor series)

- bool hierarchApproxFlag

  flags the use of a hierarchical surrogate

- bool newCenterFlag

  flags the acceptance of a candidate point and the existence of a new trust region center

- bool daceCenterPtFlag

  flags the availability of the center point in the DACE evaluations for global approximations (CCD, Box-Behnken)

- bool multiLayerBypassFlag

  flags the simultaneous presence of two conditions: (1) additional layerings w/i actual_model (e.g., surrogateModel = layered/nested/layered -> actual_model = nested/layered), and (2) a user-specification to bypass all layerings within actual_model for the evaluation of truth data (response_center_truth and response_star_truth).

- bool useGradsFlag

  flags the "use_gradients" specification in which gradients are 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

- size_t nonlinIneqOffset

  index offset to nonlinear constraint functions

- size_t nonlinEqOffset

  index offset to nonlinear constraint functions

- RealVector multiObjWts

  vector of multiobjective weights.

- RealVector nonlinIneqLowerBnds

  vector of current nonlinear inequality constraint lower bounds

- RealVector nonlinIneqUpperBnds

  vector of current nonlinear inequality constraint upper bounds

- RealVector nonlinEqTargets

  vector of current nonlinear equality constraint targets

- Real bigRealBoundSize

  cutoff value for continuous bounds

- RealVector nonlinIneqLowerBndsSlack

  vector of true nonlinear inequality constraint lower bounds

- RealVector nonlinIneqUpperBndsSlack

  vector of true nonlinear inequality constraint upper bounds

- RealVector nonlinEqTargetsSlack

  vector of true nonlinear equality constraint targets

- Real tau

  constraint relaxation parameter

- Real alpha

  constraint relaxation parameter backoff parameter (multiplier)

- int npsolDerivLevel

  derivative level for NPSOL executions (1 = analytic grads of objective fn, 2 = analytic grads of constraints, 3 = analytic grads of both).

- Variables bestVariables

  best variables found in SBO

- Response bestResponses

  best responses found in SBO

Static Private Attributes

- static SurrBasedOptStrategy ∗ sboOptInstance

  pointer to SBO strategy used in static member functions

10.97.1 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.

10.97.2 Member Function Documentation

10.97.2.1 void run_strategy () `[virtual]`

Performs the surrogate-based optimization strategy by optimizing local, global, or hierarchical surrogates over a series of trust regions.

Trust region-based strategy to perform surrogate-based optimization in subregions (trust regions) of the parameter space. The 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.

10.97.2.2 void hard_convergence_check (const Response & *response_truth*, const RealVector & *c_vars*, const RealVector & *lower_bnds*, const RealVector & *upper_bnds*) `[private]`

check for hard convergence (norm of projected gradient of merit function near zero)

The hard convergence check computes the gradient of the merit function at the trust region center, performs a projection for active bound constraints (removing any gradient component directed into an active bound), and signals convergence if the 2-norm of this projected gradient is less than convergenceTol.

10.97.2.3 void tr_ratio_check (const RealVector & *c_vars_center*, const RealVector & *c_vars_star*, const RealVector & *tr_lower_bounds*, const RealVector & *tr_upper_bounds*, const Response & *response_center_truth*, const Response & *response_center_approx*, const Response & *response_star_truth*, const Response & *response_star_approx*) `[private]`

compute trust region ratio for accepting/rejecting SBO iterate and sizing next trust region and check for soft convergence (diminishing returns)

Compute soft convergence metrics (trust region ratio, number of consecutive failures, min trust region size, etc.) and use them to assess whether the convergence rate has decreased to a point where the process should be terminated (diminishing returns).

10.97.2.4 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.

10.97.2.5 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.

10.97.2.6 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.

10.97.2.7 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.

10.97.2.8 Real lagrangian_merit (const RealVector & *fn_vals*) `[private]`

compute a Lagrangian function from a set of function values

The Lagrangian function computation sums the objective function and the Lagrange multipler terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with g<=0 and h=0.

10.97.2.9 Real augmented_lagrangian_merit (const RealVector & *fn_vals*) `[private]`

compute an augmented Lagrangian function from a set of function values

The Rockafellar augmented Lagrangian function sums the objective function, Lagrange multipler terms for inequality/equality constraints, and quadratic penalty terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with g<=0 and h=0.

10.97.2.10 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 cv.

10.97.2.11 Real objective (const RealVector & *fn_vals*) `[private]`

compute a single objective value from one or more objective functions

The objective computation sums up the contributions from one of more objective functions using the multiobjective weights.

10.97.2.12 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).

10.97.2.13 void hom_objective_eval (int & *mode*, int & *n*, double ∗ *u*, double & *f*, double ∗ *grad_f*, int &) `[static, private]`

static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.

NPSOL objective functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem .

10.97.2.14 void hom_constraint_eval (int & *mode*, int & *ncnln*, int & *n*, int & *nrowj*, int ∗ *needc*, double ∗ *u*, double ∗ *c*, double ∗ *cjac*, int & *nstate*) `[static, private]`

static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation.

NPSOL constraint functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem .

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

- SurrBasedOptStrategy.H
- SurrBasedOptStrategy.C

10.98 SurrogateDataPoint Class Reference

Container class encapsulating basic parameter and response data 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)

10.98.1 Detailed Description

Container class encapsulating basic parameter and response data for defining a "truth" data point.

A list of these data points is contained in each Approximation instance (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

10.99 SurrogateDataPointRep Class Reference

The representation of a surrogate data point. This representation, 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

10.99.1 Detailed Description

The representation of a surrogate data point. This representation, or body, may be shared by multiple SurrogateDataPoint handle instances.

The SurrogateDataPoint/SurrogateDataPointRep pairs utilize a handle-body idiom (Coplien, Advanced C++).

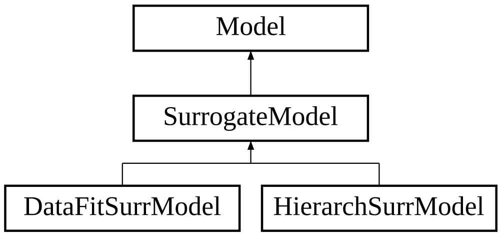
The documentation for this class was generated from the following file:

- DakotaApproximation.H

10.100 SurrogateModel Class Reference

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).

Inheritance diagram for SurrogateModel::



Protected Member Functions

- SurrogateModel (ProblemDescDB &problem_db)

  constructor

- ∼SurrogateModel ()

  destructor

- void compute_correction (const Response &truth_response, const Response &approx_response, const RealVector &c_vars)

  compute the correction required to bring approx_response into 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 check_submodel_compatibility (const Model &sub_model)

  verify compatibility between SurrogateModel attributes and attributes of the submodel (DataFitSurrModel::actualModel or HierarchSurrModel::highFidelityModel)

- bool force_rebuild ()

  evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data

- void auto_correction (bool correction_flag)

  sets autoCorrection to on (true) or off (false)

- bool auto_correction ()

  returns autoCorrection setting

Protected Attributes

- bool mixedResponseSet

  flag for mixed approximate/actual responses

- IntArray surrogateFnIds

  for mixed response sets, this array specifies the response function subset that is approximated

- ResponseArray correctedResponseArray

  array of corrected responses used in derived_synchronize() functions

- IntResponseMap correctedResponseMap

  list of corrected responses used in derived_synchronize_nowait() functions

- IntRealVectorMap rawCVarsMap

  map of raw continuous variables used by apply_correction(). Model::varsList cannot be used for this purpose since it does not contain lower level variables sets from finite differencing.

- 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

  a flag which controls the use of apply_correction() in DataFitSurrModel and HierarchSurrModel approximate response computations

- bool correctionComputed

  flag indicating whether or not a correction has been computed and is available for application

- String approxType

  approximation type identifier string: global, local, or hierarchical

- size_t approxBuilds

  number of calls to build_approximation()

- bool surrogateBypass

  a flag which allows bypassing the approximation for evaluations on the underlying truth model.

- String rebuildControl

  flag controlling the rebuild of approximations when changes occur to the active/inactive variable values/bounds.

- RealVector fitCLBnds

  stores a copy of the active continuous lower bounds when the approximation is built; used to detect when a rebuild is required.

- RealVector fitCUBnds

  stores a copy of the active continuous upper bounds when the approximation is built; used to detect when a rebuild is required.

- IntVector fitDLBnds

  stores a copy of the active discrete lower bounds when the approximation is built; used to detect when a rebuild is required.

- IntVector fitDUBnds

  stores a copy of the active discrete upper bounds when the approximation is built; used to detect when a rebuild is required.

- RealVector fitInactCVars

  stores a copy of the inactive continuous variables when the approximation is built; used to detect when a rebuild is required.

- IntVector fitInactDVars

  stores a copy of the inactive discrete variables when the approximation is built; used to detect when a rebuild is required.

- RealVector fitInactCLBnds

  stores a copy of the inactive continuous lower bounds when the approximation is built; used to detect when a rebuild is required.

- RealVector fitInactCUBnds

  stores a copy of the inactive continuous upper bounds when the approximation is built; used to detect when a rebuild is required.

- IntVector fitInactDLBnds

  stores a copy of the inactive discrete lower bounds when the approximation is built; used to detect when a rebuild is required.

- IntVector fitInactDUBnds

  stores a copy of the inactive discrete upper bounds when the approximation is built; used to detect when a 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

  flag used to indicate function values near zero for multiplicative 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

  0th-order additive correction term: equals the difference between high and low fidelity model values at x=x_center.

- RealMatrix addCorrGrads

  1st-order additive correction term: equals the gradient of the high/low function difference at x=x_center.

- RealMatrixArray addCorrHessians

  2nd-order additive correction term: equals the Hessian of the high/low function difference at x=x_center.

- RealVector multCorrFns

  0th-order multiplicative correction term: equals the ratio of high fidelity to low fidelity model values at x=x_center.

- RealMatrix multCorrGrads

  1st-order multiplicative correction term: equals the gradient of the high/low function ratio at x=x_center.

- RealMatrixArray multCorrHessians

  2nd-order multiplicative correction term: equals the Hessian of the high/low function ratio at x=x_center.

- RealVector combineFactors

  factors for combining additive and multiplicative corrections. Each factor is the weighting applied to the additive correction and 1.-factor is the weighting applied to the multiplicative correction. The factor value is determined by an additional requirement to match the high fidelity function value at the previous correction point (e.g., previous trust region center). This results in a multipoint correction instead of a strictly local correction.

- RealVector correctionCenterPt

  The point in parameter space where the current correction is calculated (often the center of the current trust region). Used in calculating (x - x_c) terms in 1st-/2nd-order corrections.

- RealVector correctionPrevCenterPt

  copy of correctionCenterPt from the previous correction cycle

- RealVector approxFnsCenter

Surrogate function values at the current correction point which are needed as a fall back if the current surrogate function values are unavailable when applying 1st-/2nd-order multiplicative corrections.

- RealVector approxFnsPrevCenter

  copy of approxFnsCenter from the previous correction cycle

- RealMatrix approxGradsCenter

  Surrogate gradient values at the current correction point which are needed as a fall back if the current surrogate function gradients are 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

10.100.1 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.

10.100.2 Member Function Documentation

10.100.2.1 void compute_correction (const Response & *truth_response*, const Response & *approx_response*, const RealVector & *c_vars*) `[protected, virtual]`

compute the correction required to bring approx_response into 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.

10.100.2.2 bool force_rebuild () `[protected]`

evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data

This function forces a rebuild of the approximation according to the approximation type, the rebuildControl setting, and which active/inactive data has changed since the last build.

10.100.3 Member Data Documentation

10.100.3.1 bool autoCorrection `[protected]`

a flag which controls the use of apply_correction() in DataFitSurrModel and HierarchSurrModel approximate response computations

SurrBasedOptStrategy must toggle this value since compute_correction() no longer automatically backs out an old correction.

10.100.3.2 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().

10.100.3.3 String rebuildControl `[protected]`

flag controlling the rebuild of approximations when changes occur to the active/inactive variable values/bounds.

A setting of "all" denotes that the approximation should be rebuilt every time the inactive variables change (e.g., for each instance of {d} in OUU). A setting of "region" denotes that the approximation should be rebuilt every time the bounded region for the inactive variables changes (e.g., for each new trust region on {d} in OUU).

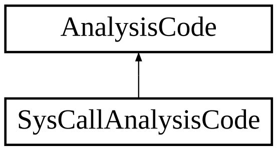
The documentation for this class was generated from the following files:

- SurrogateModel.H
- SurrogateModel.C

10.101 SysCallAnalysisCode Class Reference

Derived class in the AnalysisCode class hierarchy which spawns simulations using system calls.

Inheritance diagram for 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

  optional command usage string for supporting nonstandard command syntax (supported only by SysCall analysis codes)

10.101.1 Detailed Description

Derived class in the AnalysisCode class hierarchy which spawns simulations using system calls.

SysCallAnalysisCode creates separate simulation processes using the C system() command. It utilizes CommandShell to manage shell syntax and asynchronous invocations.

10.101.2 Member Function Documentation

10.101.2.1 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.

10.101.2.2 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.

10.101.2.3 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.

10.101.2.4 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.

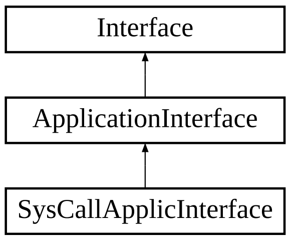
The documentation for this class was generated from the following files:

- SysCallAnalysisCode.H
- SysCallAnalysisCode.C

10.102 SysCallApplicInterface Class Reference

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

Inheritance diagram for 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)

  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.

- void derived_map_asynch (const ParamResponsePair &pair)

  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.

- void 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)

  Spawn the application by managing the input filter, analysis drivers, and output filter. Called from derived_map() & derived_map_asynch().

- void derived_synch_kernel (PRPList &prp_list)

  Convenience function for common code between derived_synch() & derived_synch_nowait().

- bool system_call_file_test (const String &root_file)

  detect completion of a function evaluation through existence of the necessary results file(s)

Private Attributes

- SysCallAnalysisCode sysCallSimulator

  SysCallAnalysisCode provides convenience functions for passing the input filter, the analysis drivers, and the output filter to a CommandShell in various combinations.

- IntSet sysCallSet

  set of function evaluation id's for active asynchronous system call evaluations

- IntShortMap failCountMap

  map linking function evaluation id's to number of response read failures

10.102.1 Detailed Description

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

SysCallApplicInterface uses a SysCallAnalysisCode object for performing simulation invocations.

10.102.2 Member Function Documentation

10.102.2.1 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 satisifies 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.

10.102.2.2 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.

10.102.2.3 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.

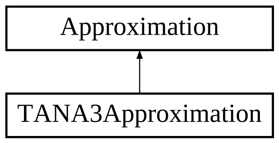
The documentation for this class was generated from the following files:

- SysCallApplicInterface.H
- SysCallApplicInterface.C

10.103 TANA3Approximation Class Reference

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

Inheritance diagram for TANA3Approximation::



Public Member Functions

- TANA3Approximation ()

  default constructor

- TANA3Approximation (ProblemDescDB &problem_db, const size_t &num_acv)

  standard constructor

- ∼TANA3Approximation ()

  destructor

Protected Member Functions

- int num_coefficients () const

  return the minimum number of samples required to build the 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

10.103.1 Detailed Description

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

The TANA3Approximation class provides a multipoint approximation based on matching value and gradient data from two points (typically the current and previous iterates) in parameter space. It forms an exponential approximation in terms of intervening variables.

10.103.2 Member Function Documentation

10.103.2.1 void clear_current () `[protected, virtual]`

Redefine default implementation to support history mechanism.

Reimplemented from Approximation.

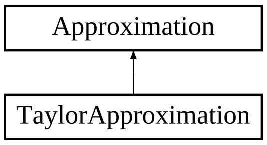
The documentation for this class was generated from the following files:

- TANA3Approximation.H
- TANA3Approximation.C

10.104 TaylorApproximation Class Reference

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

Inheritance diagram for TaylorApproximation::



Public Member Functions

- TaylorApproximation ()

  default constructor

- TaylorApproximation (ProblemDescDB &problem_db, const size_t &num_acv)

  standard constructor

- ∼TaylorApproximation ()

  destructor

Protected Member Functions

- int num_coefficients () const

  return the minimum number of samples required to build the 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

- void second_order_flag (bool flag)

  set the Approximation's secondOrderFlag, if present

Private Attributes

- bool secondOrderFlag

  flag to indicate a 2nd-order Taylor series with a Hessian term

10.104.1 Detailed Description

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

The TaylorApproximation class provides a local approximation based on data from a single point in parameter space. It uses a first- or second-order Taylor series expansion: f(x) = f(x_c) + grad(x_c)' (x - x_c) + (x - x_c)' Hess(x_c) (x - x_c) / 2.

10.104.2 Member Function Documentation

10.104.2.1 void second_order_flag (bool *flag*) `[inline, protected, virtual]`

set the Approximation's secondOrderFlag, if present

Redefined by TaylorApproximation to set secondOrderFlag.

Reimplemented from Approximation.

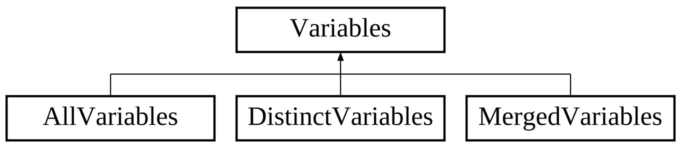
The documentation for this class was generated from the following files:

- TaylorApproximation.H
- TaylorApproximation.C

10.105 Variables Class Reference

Base class for the variables class hierarchy.

Inheritance diagram for Variables::



Public Member Functions

- Variables ()

  default constructor
- Variables (const ProblemDescDB &problem_db)

  standard constructor
- Variables (const pair< short, short > &view)

  alternate constructor
- 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 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 StringArray all_discrete_variable_labels () const

  returns a single array with all discrete variable labels

- 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 true copy is needed (the representation is _not_ shared).

- const IntList & merged_integer_list () const

  returns the list of discrete variables merged into a continuous array

- 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 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

Protected Member Functions

- Variables (BaseConstructor, const ProblemDescDB &problem_db, const pair< short, short > &view)

  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

Protected Attributes

- IntList mergedIntegerList

  the list of discrete variables for which integrality is relaxed by merging them into a continuous array

- pair< short, short > variablesView

  the variables view pair containing active (first) and inactive (second) view enumerations

- 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

  an empty real vector returned in get functions when there are no variables corresponding to the request

- IntVector emptyIntVector

  an empty int vector returned in get functions when there are no variables corresponding to the request

- StringArray emptyStringArray

  an empty label array returned in get functions when there are no variables corresponding to the request

Private Member Functions

- virtual void copy_rep (const Variables ∗vars_rep)

  Used by copy() to copy the contents of a letter class.

- Variables ∗ get_variables (const ProblemDescDB &problem_db)

Used by the standard envelope constructor to instantiate the correct letter class.

- Variables ∗ get_variables (const pair< short, short > &view) const

  Used by the alternate envelope constructor, by read functions, 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

10.105.1 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.

10.105.2 Constructor & Destructor Documentation

10.105.2.1 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.

10.105.2.2 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.

10.105.2.3 Variables (const pair< short, short > & *view*)

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_variables(view), which invokes the default constructor of the derived letter class, which in turn invokes the default constructor of the base class.

10.105.2.4 Variables (const Variables & *vars*)

copy constructor

Copy constructor manages sharing of variablesRep and incrementing of referenceCount.

10.105.2.5 ∼Variables () `[virtual]`

destructor

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

10.105.2.6 Variables (BaseConstructor, const ProblemDescDB & *problem_db*, const pair< short, short > & *view*) `[protected]`

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_variables() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_variables() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ∼Variables).

10.105.3 Member Function Documentation

10.105.3.1 Variables operator= (const Variables & *vars*)

assignment operator

Assignment operator decrements referenceCount for old variablesRep, assigns new variablesRep, and increments referenceCount for new variablesRep.

10.105.3.2 Variables copy () const

for use when a true 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).

10.105.3.3 Variables $*$ get_variables (const ProblemDescDB & *problem_db*) `[private]`

Used by the standard envelope constructor to instantiate the correct letter class.

Initializes variablesRep to the appropriate derived type, as given by problem_db attributes. The standard derived class constructors are invoked.

10.105.3.4 Variables $*$ get_variables (const pair< short, short > & *view*) const `[private]`

Used by the alternate envelope constructor, by read functions, and by copy() to instantiate a new letter class.

Initializes variablesRep to the appropriate derived type, as given by view. The default derived class constructors are invoked.

10.105.4 Member Data Documentation

10.105.4.1 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.

10.105.4.2 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.

10.105.4.3 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.

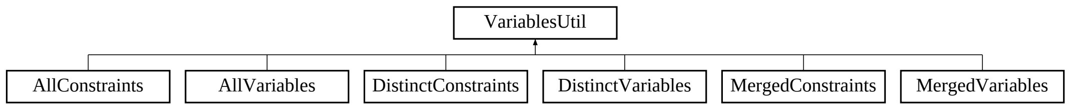
The documentation for this class was generated from the following files:

- DakotaVariables.H
- DakotaVariables.C

10.106 VariablesUtil Class Reference

Utility class for the Variables and Constraints hierarchies which provides convenience functions for variable vectors and label arrays for combining design, uncertain, and state variable types and merging continuous and discrete variable domains.

Inheritance diagram for VariablesUtil::



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

  combine a continuous array and a discrete array into a single continuous 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

  combine 3 continuous arrays (design, uncertain, state) into a single continuous array (all view)

- void update_all_discrete (const IntVector &d1_array, const IntVector &d2_array, IntVector &all_array) const

  combine 2 discrete arrays (design, state) into a single discrete array (all view)

- void update_from_merged (const RealVector &m_array, RealVector &c_array, IntVector &d_array) const

  extract a continuous array and a discrete array from a single continuous 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 *extract 3 continuous arrays (design, uncertain, state) from a single continuous array (all view)*

- void update_from_all_discrete (const IntVector &all_array, IntVector &d1_array, IntVector &d2_array) const

  extract 2 discrete arrays (design, state) from a single discrete 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 (const StringArray &l1_array, const StringArray &l2_array, const StringArray &l3_-array, StringArray &all_array) const

  combine 3 label arrays (design, uncertain, state) into a single label array (all view)

- 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

  update a portion of one label array from a portion of another label array (all view)

10.106.1 Detailed Description

Utility class for the Variables and Constraints hierarchies which provides convenience functions for variable vectors and label arrays for combining design, uncertain, and state variable types and merging continuous and discrete variable domains.

Derived classes within the Variables and Constraints hierarchies use multiple inheritance to inherit these utilities.

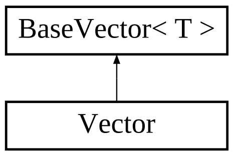
The documentation for this class was generated from the following file:

- VariablesUtil.H

10.107 Vector Class Template Reference

Template class for the Dakota numerical vector.

Inheritance diagram for 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

  Converts the Vector to a standard C-style array. 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 and associated label array in annotated from an input stream.

- 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 and the corresponding labels 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 and the corresponding labels to an output stream in aprepro format.

- void write_annotated (ostream &s, const Array< String > &label_array) const

  Writes a Vector and associated label array in annotated form to an output stream.

- 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)

  Reads a Vector and associated label array from a buffer after an 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

  Writes a Vector and associated label array to a buffer prior to an MPI send.

10.107.1 Detailed Description

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

10.107.2 Constructor & Destructor Documentation

10.107.2.1 Vector (const T $*$ *p*, size_t *len*) `[inline]`

Constructor which copies len entries from T$*$.

Assigns size values from p into array.

10.107.3 Member Function Documentation

10.107.3.1 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 11

DAKOTA File Documentation

11.1 JEGAEvaluator.C File Reference

Contains the implementation of the JEGAEvaluator class.

Namespaces

- namespace Dakota
- namespace **JEGA::Logging**
- namespace **JEGA::Algorithms**

11.1.1 Detailed Description

Contains the implementation of the JEGAEvaluator class.

11.2 JEGAEvaluator.H File Reference

Contains the definition of the JEGAEvaluator class.

Namespaces

- namespace Dakota

11.2.1 Detailed Description

Contains the definition of the JEGAEvaluator class.

11.3 JEGAOptimizer.C File Reference

Contains the implementation of the JEGAOptimizer class.

Namespaces

- namespace Dakota
- namespace **eddy::utilities**

Functions

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

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

11.3.1 Detailed Description

Contains the implementation of the JEGAOptimizer class.

11.4 JEGAOptimizer.H File Reference

Contains the definition of the JEGAOptimizer class.

Namespaces

- namespace **JEGA**
- namespace **JEGA::Utilities**
- namespace **JEGA::FrontEnd**
- namespace Dakota

11.4.1 Detailed Description

Contains the definition of the JEGAOptimizer class.

11.5 keywordtable.C File Reference

file containing keywords for the strategy, method, model, variables, interface, and responses input specifications from **dakota.input.spec**

Variables

- const struct KeywordHandler idrKeywordTable [ ]

  Initialize the keyword table as a vector of KeywordHandler structures (KeywordHandler declared in idr-keyword.h). A null KeywordHandler structure signifies the end of the keyword table.

11.5.1 Detailed Description

file containing keywords for the strategy, method, model, variables, interface, and responses input specifications from **dakota.input.spec**

11.6 main.C File Reference

file containing the main program for DAKOTA

Functions

- int main (int argc, char ∗argv[ ])

  The main DAKOTA program.

11.6.1 Detailed Description

file containing the main program for DAKOTA

11.6.2 Function Documentation

11.6.2.1 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.

11.7 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.

11.7.1 Detailed Description

file containing the DAKOTA restart utility main program

11.7.2 Function Documentation

11.7.2.1 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.

11.7.2.2 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/TECPLOT data file).

11.7.2.3 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.

11.7.2.4 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).

11.7.2.5 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.

11.7.2.6 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 12

Recommended Practices for DAKOTA Development

12.1 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.

12.2 Style Guidelines

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

12.2.1 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.:

```
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.:

```
double classMemberVariable;
```

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

```
int temporary_variable;
```

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

```
const double CONSTANT_VALUE;
```

12.2.2 Function styles

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

```
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.:

```
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.:

```
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.:

```
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
```

12.2.3 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.:

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

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

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

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

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

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

12.3 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.

12.4 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:

```
//- Class:       Model
//- Description: The model to be iterated by the Iterator.  Contains Variables, Interface, and Response objects.
//- Owner:       Mike Eldred
//- Version: $Id: Dev_Recomm_Pract.dox 3615 2006-05-10 17:39:26Z mseldre $
```

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 13

Instructions for Modifying DAKOTA's Input Specification

13.1 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/VendorPackages/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`).

13.2 Rebuild IDR

```
cd $DAKOTA/VendorPackages/idr
make clean
make
```

These steps regenerate keywordtable.C and idr-gen-code.C in the $DAKOTA/VendorPackages/idr/<canonical_build_directory> directory for use in updating keywordtable.C and IDRProblemDescDB.C in $DAKOTA/src.

13.3 Update keywordtable.C in $DAKOTA/src

Do *not* directly replace the keywordtable.C in $DAKOTA/src using the one from idr, as there are important differences in the kwhandler bindings. Rather, update the keywordtable.C in $DAKOTA/src using the one from idr as a reference. Once this step is completed, it is a good idea to verify the match by diff'ing the 2 files. The only differences should be in comments, includes, and kwhandler declarations.

13.4 Update IDRProblemDescDB.C in $DAKOTA/src

Find the keyword handler functions (e.g., variables_kwhandler()) in $DAKOTA/VendorPackages/idr/<canonical_build_directory>/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.

13.4.1 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:

```
Int cdv_descriptor = -1;
Int cdv_initial_point = -1;
```

They start after the line "Int cntr;". The keyword counter loop looks like this:

```
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.

13.4.2 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.:

```
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:

```
  [method_setting = <REAL>]
```

you would copy over

```
  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).

13.5 Update ProblemDescDB.C in $DAKOTA/src

13.5.1 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:

```
    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,

```
  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->list-Iter).attribute syntax for variables, interface, responses, and method specifications. For example, the `method_-setting` example attribute would be added to `get_drv()` as:

```
  else if (entry_name == "method.method_name.method_setting")
    return (*dbRep->dataMethodIter).methodSetting;
```

A strategy specification addition would not use a (∗dbRep->listIter) syntax, but would instead look like:

```
  else if (entry_name == "strategy.strategy_name.strategy_setting")
    return dbRep->strategySpec.strategySetting;
```

13.6 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.

13.6.1 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).

13.6.2 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.

13.7 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:

```
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.

13.8 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 14

Interfacing with DAKOTA as a Library

14.1 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 && 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` 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 instantiating the Strategy 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. This constructor assumes that MPI is initialized elsewhere in the parent application. 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.

14.2 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

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

should be replaced with its overloaded form

```
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).

14.3 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.:

```
// 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

```
problem_db.check_input();
```

will perform basic database error checking.

14.4 Instantiating the strategy

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

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

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

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

These communicators can then be used for initializing parallel simulation instances, where the number of MPI communicators in the array corresponds to one communicator per ParallelConfiguration instance, where there is one ParallelConfiguration instance per Model.

14.5 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.

14.5.1 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/DirectFnApplicInterface.[CH] for the simulation of interest. If the new function will not be a member function, then the following prototype should be used in order to pass the required data:

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

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

```
int sim();
```

since the data access can be performed through the 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.

14.5.2 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

```
namespace SIM {

class DirectFnApplicInterface: public Dakota::DirectFnApplicInterface
{
public:

  // Constructor and destructor

  DirectFnApplicInterface(const ProblemDescDB& problem_db, const size_t& num_fns);
  ~DirectFnApplicInterface();

protected:

  // Virtual function redefinitions

  int derived_map_if(const DakotaString& if_name);
  int derived_map_ac(const DakotaString& ac_name);
  int derived_map_of(const DakotaString& of_name);

private:

  // Data
}

} // namespace SIM
```

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

```
  // 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);
  // repropagate parallel configuration data down to the new interface
  first_model.reset_communicators();
```

In a more advanced case of multiple models and multiple interface plug-ins, one might use

```
  // retrieve the list of Models from the Strategy
  ModelList& models = problem_db.model_list();
  // iterate over the Model list
  for (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, num_fns), false);
      // repropagate parallel configuration data down to the new interface
      ml_iter->reset_communicators();
    }
  }
```

New 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.

14.6 Executing the strategy

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

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

14.7 Retrieving data after a run

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

```
// retrieve the final parameter values
const Variables& vars = selected_strategy.variable_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.

14.8 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 15

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.

15.1 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.

15.2 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.

15.3 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.

15.4 Todo List

Member SurfpackApproximation(ProblemDescDB &problem_db, const size_t &num_acv) The dakota data structures like RealVector inherit from std::vector.

Member SurfpackApproximation(ProblemDescDB &problem_db, const size_t &num_acv) Add RBFNet surface fit interface

Member num_coefficients() const : Check to make sure that the number of points required does not

Member num_coefficients() const : The reported number of points required is computed in a rather

Member find_coefficients() Right now, we're completely deleting the old data and then

Member approximation_coefficients() : Provide an appropriate list of coefficients for each surface type

Member get_hessian(const RealVector &x) Make this acceptably efficient

Member checkForEqualityConstraints() improve efficiency of conversion

Index