SAND2001-3514
Unlimited Release
Updated April 2003
Updated July 2004
Updated December 2004

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

Version 3.3 Developers Manual

Michael S. Eldred, Laura P. Swiler, David M. Gay, Shannon L. Brown
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

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

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, Laura P. Swiler, Steven F. Wojtkiewicz, Jr., William E. Hart, Jean-Paul Watson, David M. Gay, Shannon L. Brown

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, variables, interface, and responses keyword specifications.

The use of class hierarchies provides a clear direction 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 iteratoris 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 LayeredModel (either data fit or hierarchical). A sequence of approximate optimizations is performed, each of which involves build, optimize, and verify steps.
- NonDOptStrategy: optimization under uncertainty (OUU). Employs a single optimization iterator with a NestedModel. This NestedModel contains a sub-iterator and sub-model for performing uncertainty quantifications. In OUU approaches involving surrogates, NestedModels and LayeredModels can be chained together in a variety of ways using recursion in sub-models.
- BranchBndStrategy: mixed integer nonlinear programming using the PICO library for parallel branch and bound. Employs a single iterator with a single model, but runs multiple instances of the iterator concurrently for different variable bounds within the model.
- 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 algorithms. The Minimizer classes include:

- Optimization: Optimizer provides a base class for the DOTOptimizer, CONMINOptimizer, NPSOLOptimizer, rSQPOptimizer, and SNLLOptimizer gradient-based optimization libraries and the SGOPTOptimizer, 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 and NonDSampling. NonDSampling is then 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.
- LayeredModel: 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). LayeredModel has two derived classes: SurrLayeredModel for data fit surrogates and HierLayeredModel 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.

- FundamentalVariables: 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.
- AllMergedVariables: both variable and domain types are combined, i.e. design, uncertain, and state variable types combined (all) and continuous and discrete domain types combined (merged). The result is a single array of continuous variables.

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 FundamentalVariables view is used. For parameter studies and design of experiments, however, the variable type distinctions can be ignored and an AllVariables view is used. Finally, the branch and bound strategy relies on relaxation of integrality so that continuous optimizers may be used for mixed integer problems. In this case, a MergedVariables view is used. AllMergedVariables is included for completeness.

The VarConstraints hierarchy contains the same specializations for managing linear and bound constraints on the variables (see FundamentalVarConstraints, AllVarConstraints, MergedVarConstraints, and AllMergedVarConstraints).

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 invocation utilizes a nonblocking thread (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 allows mixing of approximation types (using the Approximation derived classes: ANNSurf, KrigingSurf, MARSSurf, RespSurf, HermiteSurf, and TaylorSurf).

Note: in the data fit approximation case, SurrLayeredModel 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 HierLayeredModel contains low and high fidelity application interfaces.

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 populates data within the ProblemDescDB support class, which maintains a DataStrategy specification and lists of DataMethod, 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, VarConstraints, Interface, 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.
- 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 Namespace Index

2.1 DAKOTA Namespace List

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

Chapter 3

DAKOTA Hierarchical Index

3.1 DAKOTA Class Hierarchy

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

Chapter 4

DAKOTA Class Index

4.1 DAKOTA Class List

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

Chapter 5

DAKOTA File Index

5.1 DAKOTA File List

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

Chapter 6

DAKOTA Page Index

6.1 DAKOTA Related Pages

Here is a list of all related documentation pages:

Chapter 7

DAKOTA Namespace Documentation

7.1

The primary namespace for DAKOTA.

Classes

- class AllMergedVarConstraints

  Derived class within the VarConstraints hierarchy which combines the all and merged data views.

- class AllMergedVariables

  Derived class within the Variables hierarchy which combines the all and merged data views.

- class AllVarConstraints

  Derived class within the VarConstraints 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 ANNSurf

  Derived approximation class for artificial neural networks.

- 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 BranchBndStrategy

  Strategy for mixed integer nonlinear programming using the PICO parallel branch and bound engine.

- 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 Analyzer

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

- class SurrogateDataPoint

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

- 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 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 VarConstraints

  Base class for the variable constraints class hierarchy.

- class Variables

  Base class for the variables class hierarchy.

- class Vector

Template class for the Dakota numerical vector.

- class DataInterface

  Container class for interface specification data.

- class DataMethod

  Container class for method 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 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 FundamentalVarConstraints

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

- class FundamentalVariables

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

- class GridApplicInterface

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

- class HermiteSurf

  Derived approximation class for Hermite polynomials (global approximation).

- class HierLayeredModel

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

- class JEGAEvaluator

  This evaluator uses Sandia National Laboratories Dakota software.

- class JEGAOptimizer

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

- class KrigingSurf

  Derived approximation class for kriging interpolation.

- class KrigApprox

  Utility class for kriging interpolation.

- class LayeredModel

  Base class for the layered models (SurrLayeredModel and HierLayeredModel).

- class MARSSurf

  Derived approximation class for multivariate adaptive regression splines.

- class MergedVarConstraints

  Derived class within the VarConstraints 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 NLSSOLLeastSq

  Wrapper class for the NLSSOL nonlinear least squares library.

- class NonDLHSSampling

Performs LHS and Monte Carlo sampling for uncertainty quantification.

- class NonDOptStrategy

  Strategy for optimization under uncertainty (robust and reliability-based design).

- class NonDPCESampling

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

- class NonDReliability

  Class for the analytical 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.

- struct BaseConstructor

  Dummy struct for overloading letter-envelope constructors.

- struct NoDBBaseConstructor

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

- class ProblemDescDB

  The database containing information parsed from the DAKOTA input file.

- class RespSurf

  Derived approximation class for polynomial regression.

- class rSQPOptimizer
- class SGOPTApplication

  Maps the evaluation functions used by SGOPT algorithms to the DAKOTA evaluation functions.

- class SGOPTOptimizer

Wrapper class for the SGOPT optimization library.

- 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 SurrBasedOptStrategy

  Strategy for provably-convergent surrogate-based optimization.

- class SurrLayeredModel

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

- 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 TaylorSurf

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

- class VariablesUtil

  Utility class for the Variables and VarConstraints 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 double **Real**
- typedef Array< Real > **RealArray**
- typedef Array< int > **IntArray**
- typedef Array< size_t > **SizetArray**
- typedef Array< String > **StringArray**
- typedef Array< StringArray > **String2DArray**
- 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< Variables > **VariablesList**
- typedef List< Response > **ResponseList**
- typedef List< Model > **ModelList**
- typedef List< Iterator > **IteratorList**
- typedef List< ParamResponsePair > **PRPList**
- typedef IntList::iterator **ILIter**
- typedef IntList::const_iterator **ILCIter**
- typedef SizetList::iterator **StLIter**
- typedef SizetList::const_iterator **StLCIter**
- typedef StringList::iterator **StringLIter**
- typedef StringList::const_iterator **StringLCIter**
- typedef VariablesList::iterator **VarsLIter**
- 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 Vector< Real > **RealVector**
- typedef Vector< int > **IntVector**
- typedef BaseVector< Real > **RealBaseVector**
- typedef Matrix< Real > **RealMatrix**
- typedef Matrix< int > **IntMatrix**
- typedef Array< RealVector > **RealVectorArray**
- typedef Array< RealVectorArray > **RealVector2DArray**
- typedef Array< RealBaseVector > **RealBaseVectorArray**
- typedef Array< RealMatrix > **RealMatrixArray**
- typedef List< RealVector > **RealVectorList**
- 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(* **Vf**)()
- typedef void(* **Calcrj**)(int *n, int *p, Real *x, int *nf, Real *r, int *ui, void *ur, Vf vf)

Enumerations

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

Functions

- bool operator== (const AllMergedVariables &vars1, const AllMergedVariables &vars2)
  equality operator
- bool operator== (const AllVariables &vars1, const AllVariables &vars2)
  equality operator
- template<> void COLINOptimizer< coliny::DIRECT >::set_rng (void)
  ——————————————————————— *Section 3* ———————————————————————
- template<> void COLINOptimizer< coliny::DIRECT >::set_method_parameters (void)
- 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::PEAreal >::set_method_parameters (void)
- template<> void COLINOptimizer< coliny::SolisWets >::set_method_parameters (void)
- CommandShell & flush (CommandShell &shell)
  convenient shell manipulator function to "flush" the shell
- template<class T> ostream & operator<< (ostream &s, const Array< T > &data)
  global ostream insertion operator for Array
- template<class T> MPIPackBuffer & operator<< (MPIPackBuffer &s, const Array< T > &data)
  global MPIPackBuffer insertion operator for Array
- template<class T> MPIUnpackBuffer & operator>> (MPIUnpackBuffer &s, Array< T > &data)
  global MPIUnpackBuffer extraction operator for Array
- 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> 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

- 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
- bool operator== (const ResponseRep &rep1, const ResponseRep &rep2)
  equality operator
- String toUpper (const String &str)
  Return upper-case version of argument.
- String toLower (const String &str)
  Return lower-case version of argument.
- 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.*
- MPIPackBuffer & operator<< (MPIPackBuffer &s, const String &data)
  Reads String from buffer.
- MPIUnpackBuffer & operator>> (MPIUnpackBuffer &s, String &data)
  Writes String to buffer.

- istream & operator>> (istream &s, VarConstraints &vc)
  istream extraction operator for VarConstraints
- ostream & operator<< (ostream &s, const VarConstraints &vc)
  ostream insertion operator for VarConstraints
- 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)
  equality operator
- 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> MPIPackBuffer & operator<< (MPIPackBuffer &s, const Vector< T > &data)

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

  global MPIUnpackBuffer extraction 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
- 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 copy_data (const Real *ptr, const int ptr_len, RealVector &drv)
  copy Real to RealVector*
- void copy_data (const Real *ptr, const int ptr_len, RealBaseVector &drbv)
  copy Real to RealBaseVector*
- void copy_data (const Real *ptr, const int nr, const int nc, RealMatrix &drm, const String &ptr_type)

  copy Real to RealMatrix*
- void copy_data (const RealMatrix &drm, Real *ptr, const int ptr_len, const String &ptr_type)
  *copy RealMatrix to Real**
- void copy_data (const Real *ptr, const int num_vec, const int vec_len, RealVectorArray &drva, const String &ptr_type)
  copy Real to RealVectorArray*
- void copy_data (const RealVector &drv, RealMatrix &drm, size_t nr, size_t nc)
  copy RealVector to RealMatrix

- void copy_data (const RealVector &drv, RealVectorArray &drva, size_t num_vec, size_t vec_len)
  copy RealVector to RealVectorArray
- void copy_data (const RealArray &dra, RealVector &drv)
  copy RealArray to RealVector
- void copy_data (const RealBaseVector &drbv, RealVector &drv)
  copy RealBaseVector to RealVector
- void copy_data (const utilib::RealVector &rv, RealVector &drv)
  copy utilib::RealVector to RealVector
- void copy_data (const RealVector &drv, utilib::RealVector &rv)
  copy RealVector to utilib::RealVector
- void copy_data (const utilib::IntVector &iv, IntVector &div)
  copy utilib::IntVector to IntVector
- void copy_data (const IntVector &div, utilib::IntVector &iv)
  copy IntVector to utilib::IntVector
- void copy_data (const utilib::IntVector &iv, IntArray &dia)
  copy utilib::IntVector to IntArray
- void copy_data (const IntList &dil, utilib::IntVector &iv)
  copy IntList to utilib::IntVector
- void copy_data (const ::ColumnVector &cv, RealBaseVector &drbv)
  copy NEWMAT::ColumnVector to RealBaseVector
- void copy_data (const RealBaseVector &drbv,::ColumnVector &cv)
  copy RealBaseVector to NEWMAT::ColumnVector
- void copy_data (const RealArray &dra,::ColumnVector &cv)
  copy RealArray to NEWMAT::ColumnVector
- void copy_data (const RealMatrix &drm,::SymmetricMatrix &sm)
  copy RealMatrix to NEWMAT::SymmetricMatrix
- void copy_data (const RealMatrix &drm,::Matrix &m)
  copy RealMatrix to NEWMAT::Matrix
- void copy_data (const TNT::Vector< Real > &tntv, RealVector &drv)
  copy TNT::Vector to RealVector
- void copy_data (const RealVector &drv, TNT::Vector< Real > &tntv)
  copy RealVector to TNT::Vector
- void copy_data (const Real *ptr, const int ptr_len, TNT::Vector< Real > &tntv)
  copy Real to TNT::Vector*

- void copy_data (const RealMatrix &drm, TNT::Matrix< Real > &tntm)
  copy RealMatrix to TNT::Matrix
- void copy_data (const Epetra_SerialDenseVector &psdv, RealVector &drv)
  copy Epetra_SerialDenseVector to RealVector
- 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 ::ColumnVector &cv, Epetra_SerialDenseVector &psdv)
  copy NEWMAT::ColumnVector to Epetra_SerialDenseVector
- void copy_data (const ::Array< DDaceSamplePoint > &dspa, RealVectorArray &drva)
  copy DDACE Array to RealVectorArray
- void copy_data (const ::Array< DDaceSamplePoint > &dspa, double *darray)
  copy DDACE Array to RealVectorArray
- 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 interface_compare (const DataInterface &di, void *search_di)
  global comparison function for DataInterface
- MPIPackBuffer & operator<< (MPIPackBuffer &s, const DataMethod &data)
  MPIPackBuffer insertion operator for DataMethod.
- MPIUnpackBuffer & operator>> (MPIUnpackBuffer &s, DataMethod &data)

MPIUnpackBuffer extraction operator for DataMethod.

- ostream & operator<< (ostream &s, const DataMethod &data)
  ostream insertion operator for DataMethod
- bool method_compare (const DataMethod &dm, void *search_dm)
  global comparison function for DataMethod
- 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
- bool responses_compare (const DataResponses &dr, void *search_dr)
  global comparison function 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
- MPIPackBuffer & operator<< (MPIPackBuffer &s, const DataVariables &data)
  MPIPackBuffer insertion operator for DataVariables.
- MPIUnpackBuffer & operator>> (MPIUnpackBuffer &s, DataVariables &data)
  MPIUnpackBuffer extraction operator for DataVariables.
- ostream & operator<< (ostream &s, const DataVariables &data)
  ostream insertion operator for DataVariables
- bool variables_compare (const DataVariables &dv, void *search_dv)
  global comparison function for DataVariables
- int salinas_main (int argc, char *argv[ ], MPI_Comm *comm)
  subroutine interface to SALINAS simulation code
- bool operator== (const FundamentalVariables &vars1, const FundamentalVariables &vars2)
  equality operator
- 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 *)
- void **calcr** (int *np, int *pp, Real *x, int *nfp, Real *r, int *ui, void *ur, Vf vf)
- void **calcj** (int *np, int *pp, Real *x, int *nfp, Real *J, int *ui, void *ur, Vf vf)
- double **rnum1** (void)
- double **rnum2** (void)
- void abort_handler (int code)

  global function which handles serial or parallel aborts

- 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)

  equality operator

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

  inequality operator

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

  search function for a particular ParamResponsePair within a List

- bool eval_id_compare (const ParamResponsePair &pair, 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

- 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

- ParallelLibrary dummy_lib (0)

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

- ProblemDescDB dummy_db (dummy_lib)

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

- Graphics dakota_graphics

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

- const int **MAXPOSDEF** = 10
- const int **NONRANDOM** = 0
- const int **RANDOM** = 1
- Dakota::GSL_Singleton **GSL_RNG**
- 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 ParallelLibrary::abort_handler())

- int mc_ptr_int = 0

  global pointer for ModelCenter API

- const int **LARGE_SCALE** = 100

7.1.1 Detailed Description

The primary namespace for DAKOTA.

The Dakota namespace encapsulates the core classes of the DAKOTA framework and prevents name clashes with third-party libraries from VendorOptimizers and VendorPackages. The C++ source files defining these core classes reside in Dakota/src as ∗.[CH].

7.1.2 Function Documentation

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

specialization of set_method_parameters() for DIRECT

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

specialization of set_method_parameters() for Cobyla

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

specialization of set_method_parameters() for APPS

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

specialization of set_runtime_parameters() for PatternSearch

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

specialization of set_method_parameters() for PatternSearch

7.1.2.6 void COLINOptimizer< coliny::PEAreal >::set_method_parameters (void)

specialization of set_method_parameters() for PEAreal

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

specialization of set_method_parameters() for SolisWets

7.1.2.8 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.

7.1.2.9 String toUpper (const String & *str*)

Return upper-case version of argument.

Returns a String converted to upper case. Calls the String upper() method.

7.1.2.10 String toLower (const String & *str*)

Return lower-case version of argument.

Returns a String converted to lower case. Calls the String lower() method.

7.1.2.11 bool operator== (const FundamentalVariables & *vars1*, const FundamentalVariables & *vars2*)

equality operator

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

7.1.2.12 bool vars_asv_compare (const ParamResponsePair & *database_pr*, void * *search_pr*) [inline]

search function for a particular ParamResponsePair within a List

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

7.1.2.13 bool eval_id_compare (const ParamResponsePair & *pair*, void * *id*) [inline]

search function for a particular ParamResponsePair within a List

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

7.1.2.14 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.

7.1.2.15 void print_restart (int *argc*, char ** *argv*, String *print_dest*)

print a restart file

Usage: "dakota_restart_util print dakota.rst"

"dakota_restart_util to_neutral dakota.rst dakota.neu"

Prints all evals. in full precision to either stdout or a neutral file. The former is useful for ensuring that duplicate detection is successful in a restarted run (e.g., starting a new method from the previous best), and the latter is used for translating binary files between platforms.

7.1.2.16 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).

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

read a restart file (neutral file format)

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

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

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

repair a restart file by removing corrupted evaluations

Usage: "dakota_restart_util remove 0.0 dakota_old.rst dakota_new.rst"

"dakota_restart_util remove_ids 2 7 13 dakota_old.rst dakota_new.rst"

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

7.1.2.19 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.

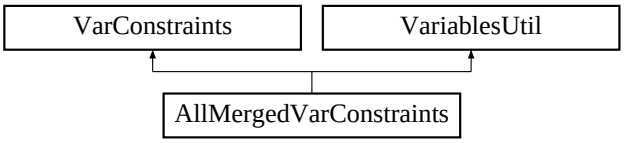
Chapter 8

DAKOTA Class Documentation

8.1 AllMergedVarConstraints Class Reference

Derived class within the VarConstraints hierarchy which combines the all and merged data views.

Inheritance diagram for AllMergedVarConstraints::



Public Member Functions

- AllMergedVarConstraints (const ProblemDescDB &problem_db)
  constructor
- ~AllMergedVarConstraints ()
  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 allMergedLowerBnds

  a continuous lower bounds array combining design, uncertain, and state variable types and merging continuous and discrete domains. The order is continuous design, discrete design, uncertain, continuous state, and discrete state.

- RealVector allMergedUpperBnds

  a continuous upper bounds array combining design, uncertain, and state variable types and merging continuous and discrete domains. The order is continuous design, discrete design, uncertain, continuous state, and discrete state.

8.1.1 Detailed Description

Derived class within the VarConstraints hierarchy which combines the all and merged data views.

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The AllMergedVarConstraints derived class combines design, uncertain, and state variable types (all) and continuous and discrete domain types (merged). The result is a single continuous lower bounds array (allMergedLowerBnds) and a single continuous upper bounds array (allMergedUpperBnds). No iterators/strategies currently use this approach; it is included for completeness and future capability.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 AllMergedVarConstraints (const ProblemDescDB & *problem_db*)

constructor

Extract fundamental variable bounds and combine them into allMergedLowerBnds and allMergedUpperBnds using utilities from VariablesUtil.

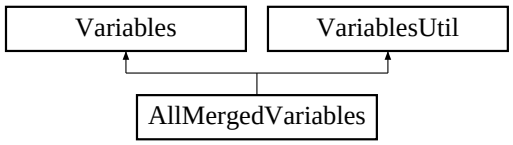
The documentation for this class was generated from the following files:

- AllMergedVarConstraints.H
- AllMergedVarConstraints.C

8.2 AllMergedVariables Class Reference

Derived class within the Variables hierarchy which combines the all and merged data views.

Inheritance diagram for AllMergedVariables::



Public Member Functions

- AllMergedVariables ()
  default constructor
- AllMergedVariables (const ProblemDescDB &problem_db)
  standard constructor
- ~AllMergedVariables ()
  destructor
- size_t tv () const
  Returns total number of vars.
- size_t cv () const
  Returns number of active continuous vars.
- size_t dv () const
  Returns number of active discrete 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
- IntVector all_discrete_variables () const
  returns a single array with all discrete variables
- 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 allMergedVars

  a continuous array combining design, uncertain, and state variable types and merging continuous and discrete domains. The order is continuous design, discrete design, uncertain, continuous state, and discrete state.

- StringArray allMergedLabels

  an array containing labels for continuous design, discrete design, uncertain, continuous state, and discrete state variables

Friends

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

  equality operator

8.2.1 Detailed Description

Derived class within the Variables hierarchy which combines the all and merged data views.

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The AllMergedVariables derived class combines design, uncertain, and state variable types (all) and continuous and discrete domain types (merged). The result is a single array of continuous variables (allMergedVars). No iterators/strategies currently use this approach; it is included for completeness and future capability.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 AllMergedVariables (const ProblemDescDB & *problem_db*)

standard constructor

Extract fundamental variable types and labels and combine them into allMergedVars and allMergedLabels using utilities from VariablesUtil.

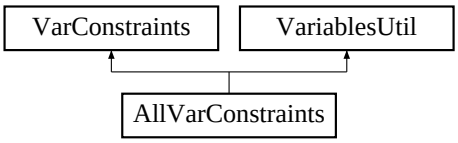
The documentation for this class was generated from the following files:

- AllMergedVariables.H
- AllMergedVariables.C

8.3 AllVarConstraints Class Reference

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

Inheritance diagram for AllVarConstraints::



Public Member Functions

- AllVarConstraints (const ProblemDescDB &problem_db)
  constructor
- ~AllVarConstraints ()
  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

8.3.1 Detailed Description

Derived class within the VarConstraints 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 AllVarConstraints 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 type selection; variables type is passed to the VarConstraints constructor in Model).

8.3.2 Constructor & Destructor Documentation

8.3.2.1 AllVarConstraints (const ProblemDescDB & *problem_db*)

constructor

Extract fundamental lower and upper bounds and combine them into allContinuousLowerBnds, allContinuousUpperBnds, allDiscreteLowerBnds, and allDiscreteUpperBnds using utilities from VariablesUtil.

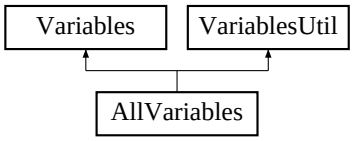
The documentation for this class was generated from the following files:

- AllVarConstraints.H
- AllVarConstraints.C

8.4 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)
  standard constructor
- ~AllVariables ()
  destructor
- size_t tv () const
  Returns total number of vars.
- size_t cv () const
  Returns number of active continuous vars.
- size_t dv () const
  Returns number of active discrete 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
- IntVector all_discrete_variables () const
  returns a single array with all discrete variables
- 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

8.4.1 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)).

8.4.2 Constructor & Destructor Documentation

8.4.2.1 AllVariables (const ProblemDescDB & *problem_db*)

standard constructor

Extract fundamental variable types and labels and combine them into allContinuousVars, allDiscreteVars, allContinuousLabels, and allDiscreteLabels using utilities from VariablesUtil.

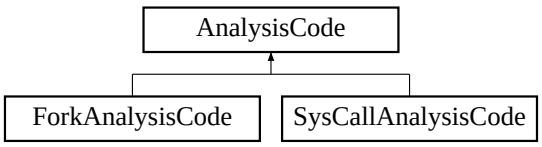
The documentation for this class was generated from the following files:

- AllVariables.H
- AllVariables.C

8.5 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_file (const Variables &vars, const IntArray &asv, const int id)
  write the variables and active set vector objects to the parameters file in either standard or aprepro format
- void read_results_file (Response &response, const int id)
  read the response object from the results file
- 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 & modified_parameters_filename () const
  return modifiedParamsFileName
- const String & modified_results_filename () const
  return modifiedResFileName
- const String & results_fname (const int id) const
  return the entry in resultsFNameList corresponding to id
- void suppress_output_flag (const bool flag)
  set suppressOutputFlag
- bool suppress_output_flag () const
  return suppressOutputFlag

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
- 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 list)
- String parametersFileName
  the name of the parameters file from user specification
- String modifiedParamsFileName
  the parameters file name actually used (modified with tagging or temp files)
- String resultsFileName
  the name of the results file from user specification
- String modifiedResFileName
  the results file name actually used (modified with tagging or temp files)

- StringList parametersFNameList

  list of parameters file names used in spawning function evaluations

- StringList resultsFNameList

  list of results file names used in spawning function evaluations

- IntList fileNameKey

  stores function evaluation identifiers to allow key-based retrieval of file names from parametersFNameList and resultsFNameList

Private Attributes

- ParallelLibrary & parallelLib

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

8.5.1 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.

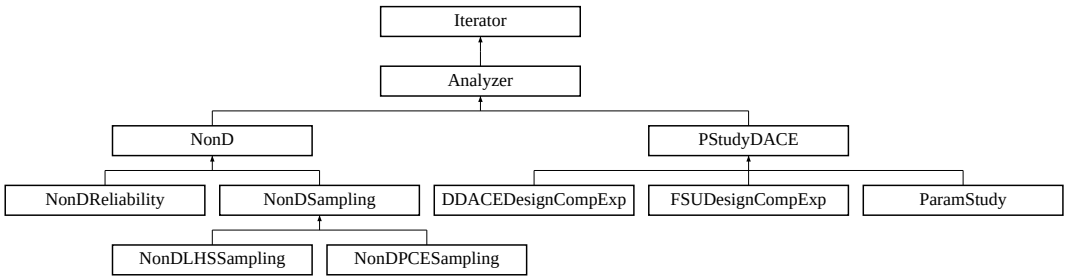
The documentation for this class was generated from the following files:

- AnalysisCode.H
- AnalysisCode.C

8.6 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 get_parameter_sets (bool vbd_change_seq_flag)

  *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

8.6.1 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.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 Analyzer (Model & *model*) [protected]

standard constructor

This constructor extracts inherited data for the optimizer and least squares branches and performs sanity checking on constraint settings.

8.6.2.2 Analyzer (NoDBBaseConstructor, Model & *model*) [protected]

alternate constructor for instantiations "on the fly"

This constructor extracts inherited data for the optimizer and least squares branches and performs sanity checking on constraint settings.

8.6.3 Member Function Documentation

8.6.3.1 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).

8.6.3.2 void var_based_decomp (const int *ndim*, const int *num_samples*) [protected]

Calculation of sensitivity indices obtained by variance based decomposition. These indices are obtained by the Saltelli version of the Sobol' VBD which uses (K+2)∗N function evaluations, where K is the number of dimensions (uncertain vars) and N is the number of samples.

8.6.3.3 void volumetric_quality (int *ndim*, int *num_samples*, double ∗ *sample_points*) [protected]

Calculation of volumetric quality measures.

Calculation of volumetric quality measures developed by FSU.

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

Printing of VBD results.

printing of variance based decomposition indices.

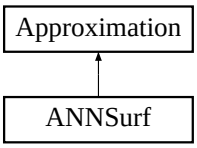
The documentation for this class was generated from the following files:

- DakotaAnalyzer.H
- DakotaAnalyzer.C

8.7 ANNSurf Class Reference

Derived approximation class for artificial neural networks.

Inheritance diagram for ANNSurf::



Public Member Functions

- ANNSurf (const ProblemDescDB &problem_db, const size_t &num_acv)

  constructor

- ~ANNSurf ()

  destructor

Protected Member Functions

- int required_samples ()

  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 the currentPoints list of SurrogateDataPoints

- Real get_value (const RealVector &x)

  retrieve the approximate function value for a given parameter vector

Private Attributes

- ANNApprox * annObject

  pointer to the ANNApprox object (see VendorPackages/ann for class declaration)

8.7.1 Detailed Description

Derived approximation class for artificial neural networks.

The ANNSurf class uses a layered-perceptron artificial neural network. Unlike most neural networks, it does not employ a back-propagation approach to training. Rather it uses a direct training approach

developed by Prof. David Zimmerman of the University of Houston and modified by Tom Paez and Chris O'Gorman of Sandia. It is more computationally efficient that back-propagation networks, but relative accuracy can be a concern.

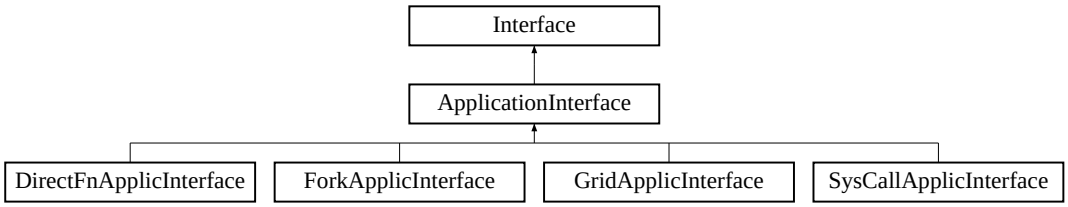
The documentation for this class was generated from the following files:

- ANNSurf.H
- ANNSurf.C

8.8 ApplicationInterface Class Reference

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

Inheritance diagram for ApplicationInterface::



Protected Member Functions

- ApplicationInterface (const ProblemDescDB &problem_db, const size_t &num_fns)

  constructor

- ~ApplicationInterface ()

  destructor

- 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 IntArray &asv, 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 IntArray &asv, 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 beforeSynchPRPList queue and returns all jobs

- const ResponseList & synch_nowait ()

  executes a nonblocking schedule for asynchronous evaluations in the beforeSynchPRPList 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 IntArray &asv, Response &response, int fn_eval_id)=0

  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)=0

  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)=0

  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)=0

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

- virtual void clear_bookkeeping ()

  clears any bookkeeping in derived classes

- 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)=0

  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

- StringArray analysisDrivers

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

- int numAnalysisDrivers

  length of analysisDrivers list

- String2DArray analysisComponents

  the set of optional analysis components used by the analysis drivers in completing a simulationwithin each function evaluation (from the analysis_drivers interface specification)

Private Member Functions

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

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

- void self_schedule_evaluations ()

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

- void static_schedule_evaluations ()

  blocking static schedule of all evaluations in beforeSynchPRPList 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 IntArray &asv, 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".

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 lenVarsASVMessage

  length of a MPIPackBuffer containing a Variables object and an active set vector 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 historyDuplicateIds

  used to bookkeep fnEvalId of asynchronous evaluations which duplicate data_pairs evaluations

- ResponseList historyDuplicateResponses

  used to bookkeep response of asynchronous evaluations which duplicate data_pairs evaluations

- IntList beforeSynchDuplicateIds

  used to bookkeep fnEvalId of asynchronous evaluations which duplicate queued beforeSynchPRPList evaluations

- SizetList beforeSynchDuplicateIndices

  used to bookkeep beforeSynchPRPList index of asynchronous evaluations which duplicate queued beforeSynchPRPList evaluations

- ResponseList beforeSynchDuplicateResponses

  used to bookkeep response of asynchronous evaluations which duplicate queued beforeSynchPRPList evaluations

- IntList runningList

used by asynchronous_local_nowait to bookkeep which jobs are running

- PRPList beforeSynchPRPList

  used to bookkeep vars/asv/response of nonduplicate asynchronous evaluations. This is the queue of jobs populated by asynchronous map() invocations which is later scheduled on a call to synch() or synch_nowait().

8.8.1 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.

8.8.2 Member Function Documentation

8.8.2.1 void init_serial () [protected, virtual]

DataInterface.C defaults of 0 servers are needed to distinguish an explicit user request for 1 server (serialization of a parallelism level) from no user request (use parallel auto-config). This default causes problems when init_communicators() is not called for an interface object (e.g., static scheduling fails in DirectFnApplicInterface::derived_map() for NestedModel::optionalInterface). This is the reason for this function: to reset certain defaults for interface objects that are used serially.

Reimplemented from Interface.

8.8.2.2 void map (const Variables & *vars*, const IntArray & *asv*, 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 beforeSynchPRPList queue for execution by one of the scheduler routines in synch() or synch_nowait(). Duplicate function evaluations are detected with duplication_detect().

Reimplemented from Interface.

8.8.2.3 const ResponseArray & synch () [protected, virtual]

executes a blocking schedule for asynchronous evaluations in the beforeSynchPRPList 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.

8.8.2.4 const ResponseList & synch_nowait () [protected, virtual]

executes a nonblocking schedule for asynchronous evaluations in the beforeSynchPRPList 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.

8.8.2.5 void serve_evaluations () [protected, virtual]

run on evaluation servers to serve the iterator master

Invoked by the serve() function in derived Model classes. Passes control to serve_evaluations_asynch(), serve_evaluations_peer(), or serve_evaluations_synch() according to specified concurrency and self/static scheduler configuration.

Reimplemented from Interface.

8.8.2.6 void stop_evaluation_servers () [protected, virtual]

used by the iterator master to terminate evaluation servers

This code is executed on the iteratorComm rank 0 processor when iteration on a particular model is complete. It sends a termination signal (tag = 0 instead of a valid fn_eval_id) to each of the slave analysis servers. NOTE: This function is called from the Strategy layer even when in serial mode. Therefore, use iteratorCommSize to provide appropriate fall through behavior.

Reimplemented from Interface.

8.8.2.7 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 com-

pleted. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

8.8.2.8 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().

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

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

Check incoming evaluation request for duplication with content of data_pairs and beforeSynchPRPList. If duplication is detected, return true, else return false. Manage bookkeeping with historyDuplicate and beforeSynchDuplicate lists. Called from map(). 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 beforeSynchPRPList search would still be meaningful. Since the intent of this request is to streamline operations, both list searches are bypassed.

8.8.2.10 void self_schedule_evaluations () [private]

blocking self-schedule of all evaluations in beforeSynchPRPList 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.

8.8.2.11 void static_schedule_evaluations () [private]

blocking static schedule of all evaluations in beforeSynchPRPList 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 beforeSynchPRPList (it runs the iterator and calls

synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

8.8.2.12 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()).

8.8.2.13 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.

8.8.2.14 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, nonblocking fork, or threads). It is called from synch_nowait() and passed the complete set of all asynchronous jobs (beforeSynchPRPList). 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 results of this function are rawResponseList and completionList. Since rawResponseList is in no particular order, completionList must be used 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().

8.8.2.15 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()).

8.8.2.16 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).

8.8.2.17 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().

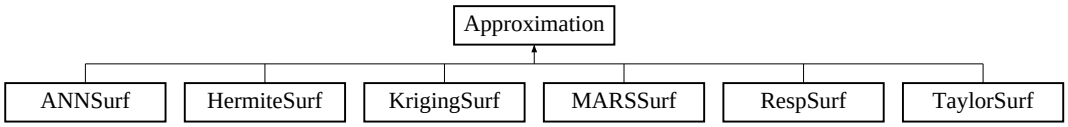
The documentation for this class was generated from the following files:

- ApplicationInterface.H
- ApplicationInterface.C

8.9 Approximation Class Reference

Base class for the approximation class hierarchy.

Inheritance diagram for Approximation::



Public Member Functions

- Approximation ()

  default constructor

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

  standard constructor for envelope

- Approximation (const Approximation &approx)

  copy constructor

- virtual ~Approximation ()

  destructor

- Approximation operator= (const Approximation &approx)

  assignment operator

- virtual 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 int required_samples ()

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

- virtual const RealVector & approximation_coefficients ()

  return the coefficient array computed by find_coefficients()

- void build (const RealVectorArray &vars_samples, const RealVector &fn_samples, const RealBaseVectorArray &grad_samples)

  build the global surface from scratch. Populates currentPoints and invokes find_coefficients().

- void build (const RealVector &vars_sample, const Real &fn_sample, const RealBaseVector &grad_-sample, const RealMatrix &hess_sample)

  build the local surface from scratch. Populates currentPoints and invokes find_coefficients().

- void add_point_rebuild (const RealVector &x, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)

  add a new point to the approximation and rebuild it

- 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).

- int num_variables () const

  return the number of variables used in the approximation

Protected Member Functions

- Approximation (BaseConstructor, const 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 the currentPoints list of SurrogateDataPoints

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 in the spec. 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

- int numCurrentPoints

  number of points in the currentPoints list

- int numSamples

number of samples passed to build() to construct the approximation

- RealBaseVector gradVector

  gradient of the approximation with respect to the variables

- RealMatrix hessMatrix

  Hessian of the approximation with respect to the variables.

- List< SurrogateDataPoint > currentPoints

  list of samples used to build the approximation

- String approxType

  approximation type (long form for diagnostic I/O)

Private Member Functions

- Approximation * get_approx (const String &approx_type, const ProblemDescDB &problem_db, const size_t &num_acv)

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

- void add_point (const RealVector &x, const Real &fn_val, const RealBaseVector &fn_grad, const RealMatrix &fn_hess)

  add a new point to the approximation (used by build & add_point_rebuild)

Private Attributes

- RealVector approxLowerBounds

  approximation lower bounds (used only by 3D graphics)

- RealVector approxUpperBounds

  approximation upper bounds (used only by 3D graphics)

- Approximation * approxRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing approxRep

8.9.1 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 a Approximation must be created for each function to be approximated (a vector of Approximations is contained in ApproximationInterface). For memory efficiency and enhanced polymorphism, the approximation hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Approximation) serves as the envelope and one of the derived classes (selected in Approximation::get_approximation()) serves as the letter.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Approximation ()

default constructor

The default constructor is used in List<Approximation> instantiations. 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.

8.9.2.2 Approximation (const String & *approx_type*, const 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.

8.9.2.3 Approximation (const Approximation & *approx*)

copy constructor

Copy constructor manages sharing of approxRep and incrementing of referenceCount.

8.9.2.4 ~Approximation () [virtual]

destructor

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

8.9.2.5 Approximation (BaseConstructor, const 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).

8.9.3 Member Function Documentation

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

assignment operator

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

8.9.3.2 Approximation * get_approx (const String & *approx_type*, const ProblemDescDB & *problem_db*, const size_t & *num_acv*) [private]

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

Used only by the envelope constructor to initialize approxRep to the appropriate derived type, as given by the approx_type parameter.

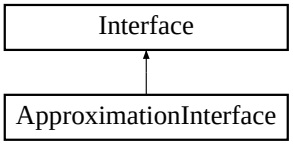
The documentation for this class was generated from the following files:

- DakotaApproximation.H
- DakotaApproximation.C

8.10 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 IntArray &asv, 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 () const

  returns minSamples

- void build_global_approximation (Iterator &dace_iterator, const RealVector &lower_bnds, const RealVector &upper_bnds)

  builds a global approximation for use as a surrogate

- void build_local_approximation (Model &actual_model)

  builds a local approximation for use as a surrogate

- void update_approximation (const RealVector &x_star, const Response &response_star)

  updates an existing global approximation with new data

- 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 ResponseList & synch_nowait ()

  recovers data from a series of asynchronous evaluations (nonblocking)

Private Attributes

- String daceMethodPointer

  string pointer to the dace iterator specified by the user in the global approximation specification

- String actualInterfacePointer

  string pointer to the actual interface specified by the user in the local/multipoint approximation specifications

- Array< Approximation > functionSurfaces

  list of approximations, one per response function

- RealVectorArray functionSurfaceCoeffs

  array of approximation coefficient vectors, one vector per response function

- 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"

- bool graphicsFlag

  controls 3D graphics of approximation surfaces

- bool useGradsFlag

  signals the use of gradient data in global approximation builds

- int minSamples

  the minimum number of samples over all functionSurfaces

- ResponseList beforeSynchResponseList

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

8.10.1 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.

8.10.2 Member Data Documentation

8.10.2.1 String daceMethodPointer [private]

string pointer to the dace iterator specified by the user in the global approximation specification

This pointer is *not* used for building objects since this is managed in SurrLayeredModels. Its use in ApproximationInterface is currently limited to flagging dace contributions to data sets in build_global_approximation().

8.10.2.2 String actualInterfacePointer [private]

string pointer to the actual interface specified by the user in the local/multipoint approximation specifications

This pointer is *not* used for building objects since this is managed in SurrLayeredModels. Its use in ApproximationInterface is currently limited to header output.

8.10.2.3 Array<Approximation> functionSurfaces [private]

list of approximations, one per response function

This formulation allows the use of mixed approximations (i.e., different approximations used for different response functions), although the input specification is not currently general enough to support it.

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

- ApproximationInterface.H
- ApproximationInterface.C

8.11 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 print (ostream &s) const

  Prints an Array to an output stream.

- void read (MPIUnpackBuffer &s)

  Reads an Array from a buffer after an MPI receive.

- void print (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.*

8.11.1 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.

8.11.2 Constructor & Destructor Documentation

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

Constructor which copies size entries from T*.

Assigns size values from p into array.

8.11.3 Member Function Documentation

8.11.3.1 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),i while for the ANSI case uses the STL assign() method.

8.11.3.2 operator T * () const [inline]

Converts the Array to a standard C-style array. Use with care!

The operator() returns a c style pointer to the data within the array. Calls the data() method. USE WITH CARE.

8.11.3.3]

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 and by SGI builds omitting exceptions (e.g., SIERRA).

8.11.3.4]

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 and by SGI builds omitting exceptions (e.g., SIERRA).

8.11.3.5 T & operator() (size_t *i*) [inline]

Index operator, not bounds checked.

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

8.11.3.6 const T & operator() (size_t *i*) const [inline]

Index operator const, not bounds checked.

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

8.11.3.7 const T * data () const [inline]

Returns pointer T* to continuous data.

Returns a C style pointer to the data within the array. USE WITH CARE. Needed to mimic RW vector class, is used in the operator(). Uses the STL front method.

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

- DakotaArray.H

8.12 BaseConstructor Struct Reference

Dummy struct for overloading letter-envelope constructors.

Public Member Functions

- BaseConstructor (int=0)

  C++ structs can have constructors.

8.12.1 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 (rather than in a header of a class that uses it) avoids problems with circular dependencies.

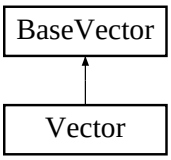
The documentation for this struct was generated from the following file:

- ProblemDescDB.H

8.13 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.

8.13.1 Detailed Description

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

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

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

8.13.2 Constructor & Destructor Documentation

8.13.2.1 BaseVector (size_t *size*, const T & *initial_val*) [inline]

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

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

8.13.3 Member Function Documentation

8.13.3.1]

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 and by SGI builds omitting exceptions (e.g., SIERRA).

8.13.3.2]

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 and by SGI builds omitting exceptions (e.g., SIERRA).

8.13.3.3 T & operator() (size_t *i*) [inline]

Index operator, not bounds checked.

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

8.13.3.4 const T & operator() (size_t *i*) const [inline]

Index operator const , not bounds checked.

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

8.13.3.5 size_t length () const [inline]

Returns size of vector.

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

8.13.3.6 void reshape (size_t *sz*) [inline]

Resizes vector to size sz.

Resizes the array to size sz by calling the STL resize method. Needed to mimic the RW vector class

8.13.3.7 const T * data () const [inline]

Returns const pointer to standard C array. Use with care.

Returns a const pointer to the data within the array. USE WITH CARE. Needed to mimic RW vector class.

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

Returns pointer to standard C array. Use with care.

Returns a non-const pointer to the data within the array. Non-const version of data() used by derived classes.

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

- DakotaBaseVector.H

8.14 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.

8.14.1 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.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 BiStream ()

Default constructor, need to open.

Default constructor, allocates xdr stream , but does not call the open method. The open method must be called before stream can be read.

8.14.2.2 BiStream (const char * *s*)

Constructor takes name of input file.

Constructor which takes a char* filename. Calls the base class open method with the filename and no other arguments. Also allocates the xdr stream.

8.14.2.3 BiStream (const char * *s*, std::ios_base::openmode *mode*)

Constructor takes name of input file, mode.

Constructor which takes a char* filename and int flags. Calls the base class open method with the filename and flags as arguments. Also allocates xdr stream.

8.14.2.4 ~BiStream ()

Destructor, calls xdr_destroy to delete xdr stream.

Destructor, destroys the xdr stream allocated in constructor

8.14.3 Member Function Documentation

8.14.3.1 BiStream & operator>> (String & *ds*)

Binary Input stream operator>>.

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

8.14.3.2 BiStream & operator>> (char * *s*)

Input operator, reads char* from binary stream BiStream.

Reading char array is a special case. The method has no way of knowing if the length to the input array is large enough, it assumes it is one char longer than actual string, (Null terminator added). As with the String the size of the xdr buffer as well as the char array size written must be read from the stream prior to reading and converting the char array.

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

- DakotaBinStream.H
- DakotaBinStream.C

8.15 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.

8.15.1 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.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 BoStream ()

Default constructor, need to open.

Default constructor allocates the xdr stream but does not call the open() method. The open() method must be called before stream can be written to.

8.15.2.2 BoStream (const char * *s*)

Constructor takes name of input file.

Constructor, takes char * filename as argument. Calls base class open method with filename and no other arguments. Also allocates xdr stream

8.15.2.3 BoStream (const char * *s*, std::ios_base::openmode *mode*)

Constructor takes name of input file, mode.

Constructor, takes char * filename and int flags as arguments. Calls base class open method with filename and flags as arguments. Also allocates xdr stream. Note : If no rpc/xdr support xdr calls are #ifdef'd out.

8.15.3 Member Function Documentation

8.15.3.1 BoStream & operator<< (const String & *ds*)

Binary Output stream operator<<.

The String operator<< must first write the xdr buffer size and the original string size to the stream. The input operator needs this information to be able to correctly read and convert the String.

8.15.3.2 BoStream & operator<< (const char * *s*)

Output operator, writes char* TO binary stream BoStream.

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

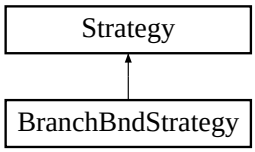
The documentation for this class was generated from the following files:

- DakotaBinStream.H
- DakotaBinStream.C

8.16 BranchBndStrategy Class Reference

Strategy for mixed integer nonlinear programming using the PICO parallel branch and bound engine.

Inheritance diagram for BranchBndStrategy::



Public Member Functions

- BranchBndStrategy (ProblemDescDB &problem_db)

  constructor

- ~BranchBndStrategy ()

  destructor

- void run_strategy ()

  Performs the branch and bound strategy by executing selectedIterator on userDefinedModel multiple times in parallel for different variable bounds within the model.

- IteratorList & iterators (bool recurse_flag=true)

  returns selectedIterator and any subordinate iterators

- ModelList & models (bool recurse_flag=true)

  returns userDefinedModel and any subordinate models

Private Attributes

- Model userDefinedModel

  the model used by the iterator

- Iterator selectedIterator

  the iterator used by BranchBndStrategy

- int numRootSamples

  number of samples to perform at the root of the branching structure

- int numNodeSamples

  number of samples to perform at each node of the branching structure

- int picoCommRank

processor rank in strategy-iterator intra comm

- int picoCommSize

  number of processors in strategy-iterator intra comm

- int argC

  dummy argument count passed to pico classes in init(), readAll(), and readAndBroadcast()

- char ** argV

  dummy argument vector passed to pico classes in init(), readAll(), and readAndBroadcast()

- utilib::DoubleVector picoLowerBnds

  global lower bounds for merged continuous & discrete design variables passed to PICO (copied from userDefinedModel)

- utilib::DoubleVector picoUpperBnds

  global upper bounds for merged continuous & discrete design variables passed to PICO (copied from userDefinedModel)

- utilib::IntVector picoListOfIntegers

  key to the discrete variables which have been relaxed and merged into the continuous variables and bounds arrays (indices in the combined arrays)

8.16.1 Detailed Description

Strategy for mixed integer nonlinear programming using the PICO parallel branch and bound engine.

This strategy combines the PICO branching engine with nonlinear programming optimizers from DAKOTA (e.g., DOT, NPSOL, OPT++) to solve mixed integer nonlinear programs. The discrete variables in the problem must support relaxation, i.e., they must be able to assume nonintegral values during the solution process. PICO selects solution "branches", each of which constrains the problem to lie within different variable bounds. The series of branches selected is designed to drive integer variables to their integral values. For each of the branches, a nonlinear DAKOTA optimizer is used to solve the optimization problem and return the solution to PICO. If this solution has all of the integer variables at integral values, then it provides an upper bound on the true solution. This bound can be used to prune other branches, since there is no need to further investigate a branch which does not yet have integral values for the integer variables and which has an objective function worse than the bound. In linear programs, the bounding and pruning processes are rigorous and will lead to the exact global optimum. In nonlinear problems, the bounding and pruning processes are heuristic, i.e. they will find local optima but the global optimum may be missed. PICO supports parallelism between "hubs," each of which drives a concurrent iterator partition in DAKOTA (and each of these iterator partitions may have lower levels of nested parallelism). This complexity is hidden from PICO through the use of picoComm, which contains the set of master iterator processors, one from each iterator partition. Thus, PICO can schedule jobs among single-processor hubs in its normal manner, unaware of the nested parallelism complexities that may occur within each nonlinear optimization.

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

- BranchBndStrategy.H
- BranchBndStrategy.C

8.17 COLINApplication Class Template Reference

Public Member Functions

- COLINApplication (Model &model, COLINOptimizerBase *opt_)

  constructor

- ~COLINApplication ()

  destructor

- void DoEval (DomainT &point, int &priority, ResponseT *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 available at construction).

Private Types

- typedef colin::OptApplication< DomainT, ResponseT > base_t

  a convenience typedef for shortening base class scoping

Private Member Functions

- void map_response (ResponseT &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

- IntArray activeSetVector

copy/conversion of the COLIN request vector

- bool dakotaModelAsynchFlag

  a flag for asynchronous DAKOTA evaluations

- ResponseList dakotaResponseList

  list of DAKOTA responses returned by synchronize_nowait()

- IntList dakotaCompletionList

  list of DAKOTA completions returned by synchronize_nowait_completions()

- size_t numObjFns

  number of objective functions

- size_t numNonlinCons

  number of nonlinear constraints

- COLINOptimizerBase * 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

- Variables * dakota_vars

  a DAKOTA variables instance used for mapping COLIN variables data

- int synchronization_state

  tracks the state of asynchronous evaluations

8.17.1 Detailed Description

template<class DomainT, class ResponseT> class Dakota::COLINApplication< DomainT, ResponseT >

COLINApplication is a DAKOTA class that is derived from COLIN's OptApplication hierarchy. It redefines a variety of virtual COLIN functions to use the corresponding DAKOTA functions. This is a more flexible algorithm library interfacing approach than can be obtained with the function pointer approaches used by NPSOLOptimizer and SNLLOptimizer.

8.17.2 Member Function Documentation

8.17.2.1 void DoEval (DomainT & *pt*, int & *priority*, ResponseT * *prob_response*, bool *synch_flag*)

launch a function evaluation either synchronously or asynchronously

Converts the DomainT 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 ResponseT response (synchronous) or bookkeeps the response object (asynchronous).

8.17.2.2 void synchronize ()

blocking retrieval of all pending jobs

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

8.17.2.3 int next_eval ()

nonblocking query and retrieval of a job if completed

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

8.17.2.4 void map_response (ResponseT & *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 ResponseT class that is compatable with COLIN.

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

- COLINApplication.H

8.18 COLINOptimizer Class Template Reference

Wrapper class for optimizers defined using COLIN.

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< ColinPoint, ColinResponse > * application

  Pointer to the COLINApplication object.

- OptProblem< ColinPoint > problem
  the COLIN problem object
- RNG * rng
  RNG ptr.
- String evalSynch
  the `synchronization` *setting (*`blocking` *or* `nonblocking`*)*

8.18.1 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.

8.18.2 Member Function Documentation

8.18.2.1 void find_optimum (void)

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.

8.18.2.2 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

8.19 ColinPoint Class Reference

Public Attributes

- vector< double > rvec
  continuous parameter values
- vector< int > ivec
  discrete parameter values

8.19.1 Detailed Description

A class containing a vector of doubles and integers.

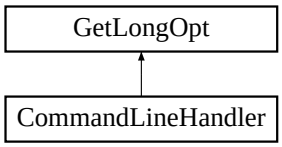
The documentation for this class was generated from the following file:

- COLINOptimizerBase.H

8.20 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.

8.20.1 Detailed Description

Utility class for managing command line inputs to DAKOTA.

CommandLineHandler provides additional functionality that is specific to DAKOTA's needs for the definition and parsing of command line options. Inheritance is used to allow the class to have all the functionality of the base class, GetLongOpt.

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

- CommandLineHandler.H
- CommandLineHandler.C

8.21 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)

8.21.1 Detailed Description

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

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

8.21.2 Member Function Documentation

8.21.2.1 CommandShell & flush ()

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

Executes the unixCommand by passing it to system(). Appends an "&" if asynchFlag is set (background system call) and echos the unixCommand to Cout if suppressOutputFlag is not set.

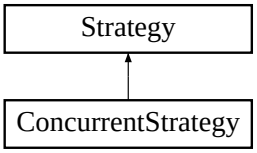
The documentation for this class was generated from the following files:

- CommandShell.H
- CommandShell.C

8.22 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.

- IteratorList & iterators (bool recurse_flag=true)

  returns selectedIterator and any subordinate iterators

- ModelList & models (bool recurse_flag=true)

  returns userDefinedModel and any subordinate models

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_strategy_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

8.22.1 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.

8.22.2 Member Function Documentation

8.22.2.1 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().

8.22.2.2 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().

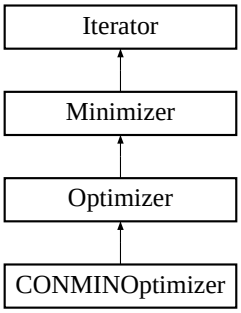
The documentation for this class was generated from the following files:

- ConcurrentStrategy.H
- ConcurrentStrategy.C

8.23 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.

Private Member Functions

- void allocate_workspace ()

  Allocates workspace for the optimizer.

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).

- RealVector localConstraintValues

  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.

- Real FDCH

  Relative finite difference step size.

- Real FDCHM

  Absolute finite difference step size.

- Real CT

  Constraint thickness parameter.

- Real CTMIN

  Minimum absolute value of CT used during optimization.

- Real CTL

  Constraint thickness parameter for linear and side constraints.

- Real CTLMIN

  Minimum value of CTL used during optimization.

- Real DELFUN

  Relative convergence criterion threshold.

- Real DABFUN

  Absolute convergence criterion threshold.

- Real * conminDesVars

  Array of design variables used by CONMIN (length N1 = numdv+2).

- Real * conminLowerBnds

  Array of lower bounds used by CONMIN (length N1 = numdv+2).

- Real * conminUpperBnds

  Array of upper bounds used by CONMIN (length N1 = numdv+2).

- Real * S

  Internal CONMIN array.

- Real * G1

  Internal CONMIN array.

- Real * G2

  Internal CONMIN array.

- Real * B

  Internal CONMIN array.

- Real * C

  Internal CONMIN array.

- int * MS1

  Internal CONMIN array.

- Real * SCAL

  Internal CONMIN array.

- Real * DF

  Internal CONMIN array.

- Real * A

  Internal CONMIN array.

- int * ISC

  Internal CONMIN array.

- int * IC

  Internal CONMIN array.

8.23.1 Detailed Description

Wrapper class for the CONMIN optimization library.

The CONMINOptimizer class provides a wrapper for CONMIN, a Public-domain Fortran 77 optimization library written by Gary Vanderplaats under contract to NASA Ames Research Center. The CONMIN User's Manual is contained in NASA Technical Memorandum X-62282, 1978. CONMIN uses a reverse communication mode, which avoids the static member function issues that arise with function pointer designs (see NPSOLOptimizer and SNLLOptimizer).

The user input mappings are as follows: `max_iterations` is mapped into CONMIN's `ITMAX` parameter, `max_function_evaluations` is implemented directly in the find_optimum() loop since there is no CONMIN parameter equivalent, `convergence_tolerance` is mapped into CONMIN's `DELFUN` and `DABFUN` parameters, `output` verbosity is mapped into CONMIN's `IPRINT` parameter (verbose: `IPRINT` = 4; quiet: `IPRINT` = 2), gradient mode is mapped into CONMIN's `NFDG` parameter, and finite difference step size is mapped into CONMIN's `FDCH` and `FDCHM` parameters. Refer to [Vanderplaats, 1978] for additional information on CONMIN parameters.

8.23.2 Member Data Documentation

8.23.2.1 int conminInfo [private]

INFO from CONMIN manual.

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

8.23.2.2 int printControl [private]

IPRINT from CONMIN manual (controls output verbosity).

Values range from 0 (nothing) to 4 (most output). 0 = nothing, 1 = initial and final function information, 2 = all of #1 plus function value and design vars at each iteration, 3 = all of #2 plus constraint values and direction vectors, 4 = all of #3 plus gradients of the objective function and constraints, 5 = all of #4 plus proposed design vector, plus objective and constraint functions from the 1-D search

8.23.2.3 int optimizationType [private]

MINMAX from DOT manual (minimize or maximize).

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

8.23.2.4 RealVector localConstraintValues [private]

array of nonlinear constraint values passed to CONMIN

This array must be of nonzero length (sized with localConstraintArraySize) and must contain only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities and equalities).

8.23.2.5 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.

8.23.2.6 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.

8.23.2.7 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.

8.23.2.8 int N1 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N1 = number of variables + 2

8.23.2.9 int N2 [private]

Size variable for CONMIN arrays. See CONMIN manual.

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

8.23.2.10 int N3 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N3 = Maximum possible number of active constraints.

8.23.2.11 int N4 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N4 = Maximum(N3,number of variables)

8.23.2.12 int N5 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N5 = 2∗(N4)

8.23.2.13 Real CT [private]

Constraint thickness parameter.

The value of CT decreases in magnitude during optimization.

8.23.2.14 Real∗ S [private]

Internal CONMIN array.

Move direction in N-dimensional space.

8.23.2.15 Real∗ G1 [private]

Internal CONMIN array.

Temporary storage of constraint values.

8.23.2.16 Real∗ G2 [private]

Internal CONMIN array.

Temporary storage of constraint values.

8.23.2.17 Real∗ B [private]

Internal CONMIN array.

Temporary storage for computations involving array S.

8.23.2.18 Real∗ C [private]

Internal CONMIN array.

Temporary storage for use with arrays B and S.

8.23.2.19 int∗ MS1 [private]

Internal CONMIN array.

Temporary storage for use with arrays B and S.

8.23.2.20 Real∗ SCAL [private]

Internal CONMIN array.

Vector of scaling parameters for design parameter values.

8.23.2.21 Real* DF [private]

Internal CONMIN array.

Temporary storage for analytic gradient data.

8.23.2.22 Real* A [private]

Internal CONMIN array.

Temporary 2-D array for storage of constraint gradients.

8.23.2.23 int* ISC [private]

Internal CONMIN array.

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

8.23.2.24 int* IC [private]

Internal CONMIN array.

Array of flags to identify active and violated constraints

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

- CONMINOptimizer.H
- CONMINOptimizer.C

8.24 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.

8.24.1 Detailed Description

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

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

- CtelRegExp.H
- CtelRegExp.C

8.25 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 interfaceType
  the interface selection: application_system/fork/direct/grid or approximation_ann/rsm/mars/hermite/ksm/mpa/taylor/hierarchical
- String idInterface
  string identifier for an interface specification data set (from the id_interface specification in InterfSetId)
- String inputFilter
  the input filter for a simulation-based interface (from the `input_filter` *specification in InterfApplic)*
- String outputFilter
  the output filter for a simulation-based interface (from the `output_filter` *specification in InterfApplic)*

- StringArray analysisDrivers

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

- String2DArray analysisComponents

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

- String parametersFile

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

- String resultsFile

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

- String analysisUsage

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

- bool apreproFormatFlag

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

- bool fileTagFlag

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

- bool fileSaveFlag

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

- int procsPerAnalysis

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

- String modelCenterFile

  configuration file for defining the simulation model accessed via the direct interface to the ModelCenter framework from Phoenix Integration (from the `modelcenter_file` *specification in InterfApplic)*

- StringArray gridHostNames

  names of host machines for a grid interface (from the `hostnames` *specification in InterfApplic)*

- IntArray gridProcsPerHost

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

- String interfaceSynchronization

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

- int asynchLocalEvalConcurrency

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

- int asynchLocalAnalysisConcurrency

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

- int evalServers

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

- 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 InterfApplic)*

- int analysisServers

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

- 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 InterfApplic)*

- String failAction

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

- int retryLimit

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

- RealVector recoveryFnVals

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

- bool activeSetVectorFlag

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

- 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 InterfApplic)*

- 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 InterfApplic)*

- String approxType

  the selected approximation type: global, multipoint, local, or hierarchical

- String actualInterfacePtr

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

- String actualInterfaceResponsesPtr

  pointer to the responses specification for constructing the truth model used in building local approximations (from the `actual_interface_responses_pointer` *specification in InterfApprox). This allows differences in gradient specifications between the responses used to build the approximation and the responses computed from the approximation.*

- String lowFidelityInterfacePtr

  pointer to the low fidelity interface specification used in hierarchical approximations (from the `low_fidelity_interface_pointer` *specification in InterfApprox)*

- String highFidelityInterfacePtr

  pointer to the high fidelity interface specification used in hierarchical approximations (from the `high_fidelity_interface_pointer` *specification in InterfApprox)*

- String approxDaceMethodPtr

  pointer to the design of experiments method used in building global approximations (from the `dace_method_pointer` *specification in InterfApprox)*

- String approxSampleReuse

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

- String approxSampleReuseFile

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

- String approxCorrectionType

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

- short approxCorrectionOrder

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

- bool approxGradUsageFlag

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

- RealVector krigingCorrelations

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

- short polynomialOrder

  scalar integer indicating the order of the polynomial approximation (1=linear, 2=quadratic, 3=cubic)

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)

8.25.1 Detailed Description

Container class for interface specification data.

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

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

- DataInterface.H
- DataInterface.C

8.26 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 methodName

  the method selection: one of the dot, npsol, opt++, apps, sgopt, nond, dace, or parameter study methods

- String idMethod

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

- String variablesPointer

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

- String interfacePointer

string pointer to the interface specification to be used by this method (from the `interface_pointer` *specification in MethodIndControl)*

- String responsesPointer

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

- String modelType

  model type selection: single, nested, or layered (from the `model_type` *specification in MethodIndControl)*

- String subMethodPointer

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

- String optionalInterfaceResponsesPointer

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

- 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 MethodIndControl)*

- 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 MethodIndControl)*

- 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 MethodIndControl)*

- 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 MethodIndControl)*

- String methodOutput

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

- 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)*

- 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 linearEqConstraintCoeffs

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

- RealVector linearEqTargets

  targets for the linear equality constraints (from the `linear_equality_targets` *specification in MethodIndControl)*

- 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, and MethodCOLINYSW*

- 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 showMiscOptions

  the `show_misc_options` *specification in MethodCOLINYDC*

- StringArray miscOptions

  the `misc_options` *specification in MethodCOLINYDC*

- Real solnAccuracy

  the `solution_accuracy` *specification in MethodSGOPTDC*

- Real crossoverRate

  the `crossover_rate` *specification for GA/EPSA methods in MethodSGOPTEA*

- Real mutationDimRate

  the `dimension_rate` *specification for mutation in GA/EPSA methods in MethodSGOPTEA*

- Real mutationPopRate

  the `population_rate` *specification for mutation in GA/EPSA methods in MethodSGOPTEA*

- Real mutationScale

  the `mutation_scale` *specification for GA/EPSA methods in MethodSGOPTEA*

- Real mutationMinScale

  the `min_scale` *specification for mutation in EPSA methods in MethodSGOPTEA*

- Real initDelta

  the `initial_delta` *specification for APPS/PS/SW methods in MethodCOLINYAPPS, MethodSGOPTPS, and MethodSGOPTSW*

- Real threshDelta

  the `threshold_delta` *specification for APPS/PS/SW methods in MethodCOLINYAPPS, MethodSGOPTPS, and MethodSGOPTSW*

- Real contractFactor

  the `contraction_factor` *specification for APPS/PS/SW methods in MethodCOLINYAPPS, MethodSGOPTPS, and MethodSGOPTSW*

- int newSolnsGenerated

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

- int numberRetained

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

- bool expansionFlag

  the `no_expansion` *specification for APPS/PS/SW methods in MethodCOLINYAPPS, MethodSGOPTPS, and MethodSGOPTSW*

- int expandAfterSuccess

  the `expand_after_success` *specification for PS/SW methods in MethodSGOPTPS and MethodSGOPTSW*

- int contractAfterFail

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

- int mutationRange

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

- int numPartitions

  the `num_partitions` *specification for EPSA methods in MethodSGOPTEA*

- int totalPatternSize

  the `total_pattern_size` *specification for APPS/PS methods in MethodCOLINYAPPS and MethodSGOPTPS*

- int batchSize

  the `batch_size` *specification for the sMC method in MethodSGOPTSMC*

- bool nonAdaptiveFlag

  the `non_adaptive` *specification for the pga_real method in MethodSGOPTEA*

- bool randomizeOrderFlag

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

- String selectionPressure

  the `selection_pressure` *specification for GA/EPSA methods in MethodSGOPTEA*

- String replacementType

  the `replacement_type` *specification for GA/EPSA methods in MethodSGOPTEA*

- String crossoverType

  the `crossover_type` *specification for GA/EPSA methods in MethodSGOPTEA*

- String mutationType

  the `mutation_type` *specification for GA/EPSA methods in MethodSGOPTEA*

- String exploratoryMoves

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

- String patternBasis

  the `pattern_basis` *specification for APPS/PS methods in MethodCOLINYAPPS and MethodSGOPTPS*

- IntArray varPartitions

  the `partitions` *specification for sMC/PStudy methods in MethodSGOPTSMC and MethodPSMPS*

- 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.

- size_t dominationCutoff

  The cutoff value for survival in domination count selection.

- Real shrinkagePercent

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

- 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 exteriorPenaltyMultiplier

  The penalty multiplier to use with penalty fitness assessors.

- String initializationType

  The means by which the JEGA should initialize the population.

- String flatFile

  The filename to use for initialization.

- int populationSize

  the `population_size` *specification for GA methods in MethodSGOPTEA, MethodCOLINY, and*

- 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 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 SGOPT, 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_linearize_mean`, `x_linearize_mpp`, `u_linearize_mean`, `u_linearize_mpp`, *or* `no_linearize` *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, MethodNonDPCE, 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*

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)

8.26.1 Detailed Description

Container class for method specification data.

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

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

- DataMethod.H
- DataMethod.C

8.27 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 multiObjectiveWeights

  vector of multiobjective weightings (from the `multi_objective_weights` *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 nonlinearEqTargets

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

8.27.1 Detailed Description

Container class for responses specification data.

The DataResponses class is used to contain the data from a responses keyword specification. It is populated by ProblemDescDB::responses_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataResponses objects is maintained in ProblemDescDB::responsesList, one for each responses 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

8.28 DataStrategy Class Reference

Container class for strategy specification data.

Public Member Functions

- DataStrategy ()

  constructor

- DataStrategy (const DataStrategy &)

  copy constructor

- ~DataStrategy ()

  destructor

- DataStrategy & operator= (const DataStrategy &)

  assignment operator

- void write (ostream &s) const

  write a DataStrategy object to an ostream

- void read (MPIUnpackBuffer &s)

  read a DataStrategy object from a packed MPI buffer

- void write (MPIPackBuffer &s) const

  write a DataStrategy object to a packed MPI buffer

Public Attributes

- String strategyType

  the strategy selection: multi_level, surrogate_based_opt, opt_under_uncertainty, 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, StratOUU, StratBandB, and StratParetoSet and method_pointer specifications in StratSingle and StratMultiStart)

- int branchBndNumSamplesRoot

  number of samples at the root for the branch and bound strategy (from the num_samples_at_root specification in StratBandB)

- int branchBndNumSamplesNode

  number of samples at each node for the branch and bound strategy (from the num_samples_at_node specification in StratBandB)

- 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_region_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 fcn) in the surrogate-based optimization strategy (from the expand_region_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)

- 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)

8.28.1 Detailed Description

Container class for strategy specification data.

The DataStrategy class is used to contain the data from a strategy keyword specification. It is populated by ProblemDescDB::strategy_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. 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

8.29 DataVariables Class Reference

Container class for variables specification data.

Public Member Functions

- DataVariables ()
  constructor
- DataVariables (const DataVariables &)
  copy constructor
- ~DataVariables ()
  destructor
- DataVariables & operator= (const DataVariables &)
  assignment operator
- bool operator== (const DataVariables &)
  equality operator
- void write (ostream &s) const
  write a DataVariables object to an ostream
- void read (MPIUnpackBuffer &s)
  read a DataVariables object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  write a DataVariables object to a packed MPI buffer
- size_t design ()
  return total number of design variables
- size_t uncertain ()
  return total number of uncertain variables
- size_t state ()
  return total number of state variables
- size_t num_continuous_variables ()
  return total number of continuous variables
- 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 numWeibullUncVars

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

- size_t numHistogramUncVars

  number of histogram uncertain variables (from the `histogram_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)*

- 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 normalUncDistLowerBnds

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

- RealVector normalUncDistUpperBnds

  distribution upper bounds for the normal uncertain variables (from the `nuv_dist_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 lognormalUncDistLowerBnds

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

- RealVector lognormalUncDistUpperBnds

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

- RealVector uniformUncDistLowerBnds

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

- RealVector uniformUncDistUpperBnds

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

- RealVector loguniformUncDistLowerBnds

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

- RealVector loguniformUncDistUpperBnds

  distribution upper bounds for the loguniform uncertain variables (from the `luuv_dist_upper_bounds` *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)*

- RealVector weibullUncDistLowerBnds

  distribution lower bounds for the weibull uncertain variables (from the `wuv_dist_lower_bounds` *specification in VarUV)*

- RealVector weibullUncDistUpperBnds

  distribution upper bounds for the weibull uncertain variables (from the `wuv_dist_upper_bounds` *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)*

- 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 uncertainDistLowerBnds

  distribution lower bounds for all uncertain variables (collected from `nuv_dist_lower_bounds`, `lnuv_dist_lower_bounds`, `uuv_dist_lower_bounds`, `luuv_dist_lower_bounds`, `wuv_dist_lower_bounds`, *and* `huv_dist_lower_bounds` *specifications in VarUV)*

- RealVector uncertainDistUpperBnds

  distribution upper bounds for all uncertain variables (collected from `nuv_dist_upper_bounds`, `lnuv_dist_upper_bounds`, `uuv_dist_upper_bounds`, `luuv_dist_upper_bounds`, `wuv_dist_upper_bounds`, *and* `huv_dist_upper_bounds` *specifications in VarUV)*

- StringArray uncertainLabels

labels for all uncertain variables (collected from `nuv_descriptors`, `lnuv_descriptors`, `uuv_descriptors`, `luuv_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)

8.29.1 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).

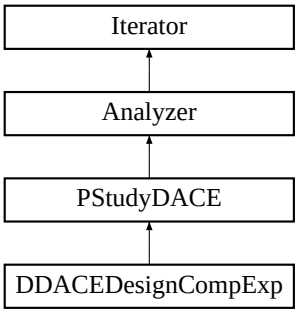
The documentation for this class was generated from the following files:

- DataVariables.H
- DataVariables.C

8.30 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 get_parameter_sets (bool vbd_change_seq_flag)

  *Returns one block of samples (ndim * num_samples).*

Private Member Functions

- 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 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 executions of run_iterator() for this object

- bool varyPattern

  flag for continuing the random number sequence from a previous run_iterator() 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

8.30.1 Detailed Description

Wrapper class for the DDACE design of experiments library.

The DDACEDesignCompExp class provides a wrapper for DDACE, a C++ design of experiments library from the Computational Sciences and Mathematics Research (CSMR) department at Sandia's Livermore CA site. This class uses design and analysis of computer experiments (DACE) methods to sample the design space spanned by the bounds of a Model. It returns all generated samples and their corresponding responses as well as the best sample found.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 DDACEDesignCompExp (Model & *model*)

primary constructor for building a standard DACE iterator

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

8.30.3 Member Function Documentation

8.30.3.1 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.

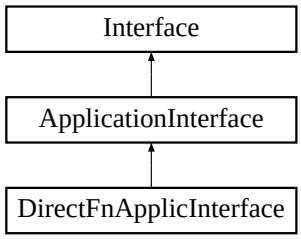
The documentation for this class was generated from the following files:

- DDACEDesignCompExp.H
- DDACEDesignCompExp.C

8.31 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, const size_t &num_fns)

  constructor

- ~DirectFnApplicInterface ()

  destructor

- void derived_map (const Variables &vars, const IntArray &asv, 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)

  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 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 ()

  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

- String pxcFile

  name of the ModelCenter simulation config file

- 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 numGradVars

  number of active continuous variables

- RealVector xC

  continuous variable set used within direct simulator functions

- IntVector xD

  discrete variable set used within direct simulator functions

- RealVector fnVals
  response function values set within direct simulator functions
- RealMatrix fnGrads
  response function gradients set within direct simulator functions
- RealMatrixArray fnHessians
  response function Hessians set within direct simulator functions
- Variables directFnVars
  class scope variables object
- IntArray directFnASV
  class scope active set vector object
- Response directFnResponse
  class scope response object

Private Member Functions

- int cantilever (const Variables &vars, const IntArray &asv, Response &response)
  the cantilever optimization under uncertainty test function
- int cyl_head (const Variables &vars, const IntArray &asv, Response &response)
  the cylinder head constrained optimization test function
- int rosenbrock (const Variables &vars, const IntArray &asv, Response &response)
  the rosenbrock optimization and least squares test function
- int text_book (const Variables &vars, const IntArray &asv, Response &response)
  the text_book constrained optimization test function
- int text_book1 (const Variables &vars, const IntArray &asv, Response &response)
  portion of text_book() evaluating the objective function and its derivatives
- int text_book2 (const Variables &vars, const IntArray &asv, Response &response)
  portion of text_book() evaluating constraint 1 and its derivatives
- int text_book3 (const Variables &vars, const IntArray &asv, Response &response)
  portion of text_book() evaluating constraint 2 and its derivatives
- int text_book_ouu (const Variables &vars, const IntArray &asv, Response &response)
  the text_book_ouu optimization under uncertainty test function
- int log_ratio (const Variables &vars, const IntArray &asv, Response &response)
  the log_ratio uncertainty quantification test function

- int short_column (const Variables &vars, const IntArray &asv, Response &response)
  the short_column uncertainty quantification/optimization under uncertainty test function
- int salinas (const Variables &vars, const IntArray &asv, Response &response)
  direct interface to the SALINAS structural dynamics simulation code
- int mc_api_run (const Variables &vars, const IntArray &asv, Response &response)
  Call ModelCenter via API, HKIM 4/3/03.

8.31.1 Detailed Description

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

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

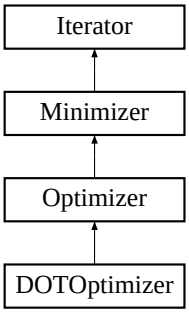
The documentation for this class was generated from the following files:

- DirectFnApplicInterface.H
- DirectFnApplicInterface.C

8.32 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.

Private Member Functions

- void allocate_workspace ()

  Allocates workspace for the optimizer.

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 localConstraintValues

  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.

8.32.1 Detailed Description

Wrapper class for the DOT optimization library.

The DOTOptimizer class provides a wrapper for DOT, a commercial Fortran 77 optimization library from Vanderplaats Research and Development. It uses a reverse communication mode, which avoids the static member function issues that arise with function pointer designs (see NPSOLOptimizer and SNLLOptimizer).

The user input mappings are as follows: `max_iterations` is mapped into DOT's `ITMAX` parameter within its `IPRM` array, `max_function_evaluations` is implemented directly in the find_optimum() loop since there is no DOT parameter equivalent, `convergence_tolerance` is mapped into DOT's

DELOBJ parameter (the relative convergence tolerance) within its RPRM array, output verbosity is mapped into DOT's IPRINT parameter within its function call parameter list (verbose: IPRINT = 7; quiet: IPRINT = 3), and optimization_type is mapped into DOT's MINMAX parameter within its function call parameter list. Refer to [Vanderplaats Research and Development, 1995] for information on IPRM, RPRM, and the DOT function call parameter list.

8.32.2 Member Data Documentation

8.32.2.1 int dotInfo [private]

INFO from DOT manual.

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

8.32.2.2 int dotFDSinfo [private]

internal DOT parameter NGOTOZ

the DOT parameter list has been modified to pass NGOTOZ, which signals whether DOT is finite-differencing (nonzero value) or performing the line search (zero value).

8.32.2.3 int dotMethod [private]

METHOD from DOT manual.

For nonlinear constraints: 0/1 = dot_mmfd, 2 = dot_slp, 3 = dot_sqp. For unconstrained: 0/1 = dot_bfgs, 2 = dot_frcg.

8.32.2.4 int printControl [private]

IPRINT from DOT manual (controls output verbosity).

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

8.32.2.5 int optimizationType [private]

MINMAX from DOT manual (minimize or maximize).

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

8.32.2.6 RealArray realCntlParmArray [private]

RPRM from DOT manual.

Array of real control parameters.

8.32.2.7 IntArray intCntlParmArray [private]

IPRM from DOT manual.

Array of integer control parameters.

8.32.2.8 RealVector localConstraintValues [private]

array of nonlinear constraint values passed to DOT

This array must be of nonzero length (sized with localConstraintArraySize) and must contain only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities and equalities).

8.32.2.9 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.

8.32.2.10 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.

8.32.2.11 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

8.33 ErrorTable Struct Reference

Data structure to hold errors.

Public Attributes

- CtelRegexp::RStatus rc

  Enumerated type to hold status codes.

- const char * msg

  Holds character string error message.

8.33.1 Detailed Description

Data structure to hold errors.

This module implements a C++ wrapper for Regular Expressions based on the public domain engine for regular expressions released by: Copyright (c) 1986 by University of Toronto. Written by Henry Spencer. Not derived from licensed software.

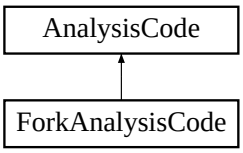
The documentation for this struct was generated from the following file:

- CtelRegExp.C

8.34 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 argument_list (const int index, const String &arg)

  set argList[index] to arg

- void tag_argument_list (const int index, const int tag)

  append an additional tag to argList[index] (beyond that already present in the modified file names) for managing concurrent analyses within a function evaluation

Private Attributes

- const char * argList [4]

  *an array of strings for use with execvp(const char *, char * const *) (an argList entry can be passed as the first argument, and the entire argList can be cast as the second argument)*

8.34.1 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.

8.34.2 Member Function Documentation

8.34.2.1 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 3-piece interface terminated abnormally (WIFEXITED(status)==0) or if either execvp or the application returned a status code of -1 (WIFEXITED(status)!=0 && (signed char)WEXITSTATUS(status)==-1). If one of these conditions is detected, output a failure message and abort. Note: the application code should not return a status code of -1 unless an immediate abort of dakota is wanted. If for instance, failure capturing is to be used, the application code should write the word "FAIL" to the appropriate results file and return a status code of 0 through exit().

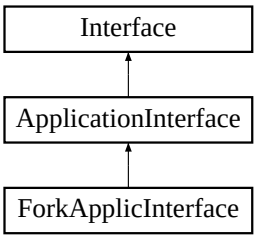
The documentation for this class was generated from the following files:

- ForkAnalysisCode.H
- ForkAnalysisCode.C

8.35 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, const size_t &num_fns)

  constructor

- ~ForkApplicInterface ()

  destructor

- void derived_map (const Variables &vars, const IntArray &asv, 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)

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

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 assignments asynchronously

Private Attributes

- ForkAnalysisCode forkSimulator

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

- List< pid_t > processIdList

  list of process id's for asynchronous evaluations; correspondence to evalIdList used for mapping captured fork process id's to function evaluation id's

- IntList evalIdList

  list of function evaluation id's for asynchronous evaluations; correspondence to processIdList used for mapping captured fork process id's to function evaluation id's

8.35.1 Detailed Description

Derived application interface class which spawns simulation codes using forks.

ForkApplicInterface uses a ForkAnalysisCode object for performing simulation invocations.

8.35.2 Member Function Documentation

8.35.2.1 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.

8.35.2.2 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.

8.35.2.3 void synchronous_local_analyses (const int & *start*, const int & *end*, const int & *step*) [private]

execute analyses synchronously on the local processor

Execute analyses synchronously in succession on the local processor (start to end in step increments). Modeled after ApplicationInterface::synchronous_local_evaluations().

8.35.2.4 void serve_analyses_asynch () [private]

serve the analysis scheduler and execute analysis assignments 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.

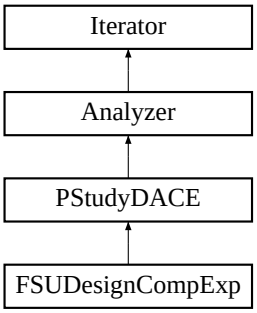
The documentation for this class was generated from the following files:

- ForkApplicInterface.H
- ForkApplicInterface.C

8.36 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 get_parameter_sets (bool vbd_change_seq_flag)

  *Returns one block of samples (ndim * num_samples).*

- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)

  reset sampling iterator

- const String & sampling_scheme () const

  return sampling name

Private Member Functions

- void enforce_input_rules ()

  enforce sanity checks/modifications for the user input specification

Private Attributes

- 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 executions of run_iterator() 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_iterator() 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.

8.36.1 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 Tessela-tion (CVT) methods to uniformly sample the parameter space spanned by the active bounds of the current Model. It returns all generated samples and their corresponding responses as well as the best sample found.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 FSUDesignCompExp (Model & *model*)

primary constructor for building a standard DACE iterator

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

8.36.3 Member Function Documentation

8.36.3.1 void enforce_input_rules () [private]

enforce sanity checks/modifications for the user input specification

Users may input a variety of quantities, but this function must enforce any restrictions imposed by the sampling algorithms.

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

- FSUDesignCompExp.H
- FSUDesignCompExp.C

8.37 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 testFunction.

Private Attributes

- bool(∗ testFunction)(const T &, void ∗)
  Pointer to test function.
- void ∗ search_val
  Holds the value to search for.

8.37.1 Detailed Description

template<class T> class Dakota::FunctionCompare< T >

Internal functor to mimic the RW find and index functions using the STL find_if() method. The class holds a pointer to the test function and the search value.

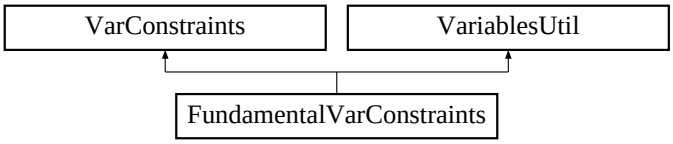
The documentation for this class was generated from the following file:

- DakotaList.H

8.38 FundamentalVarConstraints Class Reference

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

Inheritance diagram for FundamentalVarConstraints::



Public Member Functions

- FundamentalVarConstraints (const ProblemDescDB &problem_db)
  constructor
- ~FundamentalVarConstraints ()
  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

- bool nonDFlag
  this flag is set if uncertain variables are active (the default is design variables are active; see constructor for logic)
- 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 uncertainDistLowerBnds

  the uncertain distribution lower bounds array

- RealVector uncertainDistUpperBnds

  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

8.38.1 Detailed Description

Derived class within the VarConstraints 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 FundamentalVarConstraints 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, Merged, or AllMerged views use this approach (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to the VarConstraints constructor in Model).

8.38.2 Constructor & Destructor Documentation

8.38.2.1 FundamentalVarConstraints (const ProblemDescDB & *problem_db*)

constructor

Extract fundamental lower and upper bounds (VariablesUtil is not used).

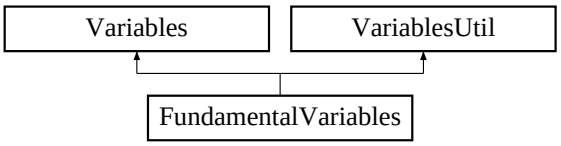
The documentation for this class was generated from the following files:

- FundamentalVarConstraints.H
- FundamentalVarConstraints.C

8.39 FundamentalVariables Class Reference

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

Inheritance diagram for FundamentalVariables::



Public Member Functions

- FundamentalVariables ()
  default constructor
- FundamentalVariables (const ProblemDescDB &problem_db)
  standard constructor
- ~FundamentalVariables ()
  destructor
- size_t tv () const
  Returns total number of vars.
- size_t cv () const
  Returns number of active continuous vars.
- size_t dv () const
  Returns number of active discrete 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
- IntVector all_discrete_variables () const
  returns a single array with all discrete variables
- 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

- bool nonDFlag

  this flag is set if uncertain variables are active (the default is design variables are active; see constructor for logic)

- 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 FundamentalVariables &vars1, const FundamentalVariables &vars2)

  equality operator

8.39.1 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 FundamentalVariables 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, Merged, or AllMerged views use this approach (see Variables::get_variables(problem_db)).

8.39.2 Constructor & Destructor Documentation

8.39.2.1 FundamentalVariables (const ProblemDescDB & *problem_db*)

standard constructor

Extract fundamental variable types and labels (VariablesUtil is not used).

8.39.3 Friends And Related Function Documentation

8.39.3.1 bool operator== (const FundamentalVariables & *vars1*, const FundamentalVariables & *vars2*) [friend]

equality operator

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

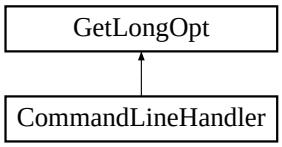
The documentation for this class was generated from the following files:

- FundamentalVariables.H
- FundamentalVariables.C

8.40 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

8.40.1 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 "=".

8.40.2 Constructor & Destructor Documentation

8.40.2.1 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.

8.40.3 Member Function Documentation

8.40.3.1 int parse (int *argc*, char ∗const ∗ *argv*)

parse the command line args (argc, argv).

A return value < 1 represents a parse error. Appropriate error messages are printed when errors are seen. parse returns the the optind (see getopt(3)) if parsing is successful.

8.40.3.2 int parse (char ∗const *str*, char ∗const *p*)

parse a string of options (typically given from the environment).

A return value < 1 represents a parse error. Appropriate error messages are printed when errors are seen. parse takes two strings: the first one is the string to be parsed and the second one is a string to be prefixed to the parse errors.

8.40.3.3 int enroll (const char ∗const *opt*, const OptType *t*, const char ∗const *desc*, const char ∗const *val*)

Add an option to the list of valid command options.

enroll adds option specifications to its internal database. The first argument is the option sting. The second is an enum saying if the option is a flag (Valueless), if it requires a mandatory value (MandatoryValue) or if it takes an optional value (OptionalValue). The third argument is a string giving a brief description of the option. This description will be used by GetLongOpt::usage. GetLongOpt, for usage-printing, uses {$val} to represent values needed by the options. {<$val>} is a mandatory value and {[$val]} is an optional value. The final argument to enroll is the default string to be returned if the option is not specified. For flags (options with Valueless), use "" (empty string, or in fact any arbitrary string) for specifying TRUE and 0 (null pointer) to specify FALSE.

8.40.3.4 const char ∗ retrieve (const char ∗const *opt*) const

Retrieve value of option.

The values of the options that are enrolled in the database can be retrieved using retrieve. This returns a string and this string should be converted to whatever type you want. See atoi, atof, atol, etc. If a "parse" is not done before retrieving all you will get are the default values you gave while enrolling! Ambiguities while retrieving (may happen when options are abbreviated) are resolved by taking the matching option that was enrolled last. For example, -{v} will expand to {-verify}. If you try to retrieve something you didn't enroll, you will get a warning message.

8.40.3.5 void usage (const char ∗ *str*) [inline]

Change header of usage output to str.

GetLongOpt::usage is overloaded. If passed a string "str", it sets the internal usage string to "str". Otherwise it simply prints the command usage.

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

- CommandLineHandler.H
- CommandLineHandler.C

8.41 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

- void tabular_counter_label (const String &label)

  set tabularCntrLabel equal to label

Private Attributes

- Graphics2D * graphics2D

  pointer to the 2D graphics object

- bool win2dOn

  flag to indicate if 2D graphics window is active

- bool win3dOn

  flag to indicate if 3D graphics window is active

- bool tabularDataFlag

  flag to indicate if tabular data stream is active

- int graphicsCntr

  used for x axis values in 2D graphics and for 1st column in tabular data

- String tabularCntrLabel

  label for counter used in first line comment w/i the tabular data file

- ofstream tabularDataFStream

  file stream for tabulation of graphics data within compute_response

8.41.1 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).

8.41.2 Member Function Documentation

8.41.2.1 void create_plots_2d (const Variables & *vars*, const Response & *response*)

creates the 2d graphics window and initializes the plots

Sets up a single event loop for duration of the dakotaGraphics object, continuously adding data to a single window. There is no reset. To start over with a new data set, you need a new object (delete old and instantiate new).

8.41.2.2 void create_tabular_datastream (const Variables & *vars*, const Response & *response*, const String & *tabular_data_file*)

opens the tabular data file stream and prints the headings

Opens the tabular data file stream and prints headings, one for each continuous and discrete variable and one for each response function, using the variable and response function labels. This tabular data is used for post-processing of DAKOTA results in Matlab, Tecplot, etc.

8.41.2.3 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.

8.41.2.4 void add_datapoint (int *i*, double *x*, double *y*)

adds data to a single window in the 2d graphics

Adds data to a single 2d plot. Allows complete flexibility in defining other kinds of x-y plotting in the 2D graphics.

8.41.2.5 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.

8.41.2.6 void show_data_3d (const RealVector & *X*, const RealVector & *Y*, const RealMatrix & *F*)

generate a new 3d plot for F(X,Y)

3D plotting clears data set and builds from scratch each time show_data3d is called. This still involves an event loop waiting for a mouse click (right button) to continue. X = 1-D x grid values only and Y = 1-D Y grid values only [X and Y are _not_ (X,Y) pairs]. F = 2-d grid of values for a single function for all (X,Y) combinations.

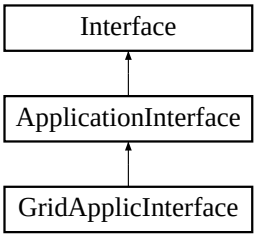
The documentation for this class was generated from the following files:

- DakotaGraphics.H
- DakotaGraphics.C

8.42 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, const size_t &num_fns)

  constructor

- ~GridApplicInterface ()

  destructor

- void derived_map (const Variables &vars, const IntArray &asv, 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)

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

Private Member Functions

- XMLObject getXML (const Variables &vars)

  convert Variables -> XMLObject

- Response getResponse (const XMLObject &xml)

  convert XMLObject -> Variables

Private Attributes

- StringArray hostNames

  array of host names to execute remote jobs

- IntArray procsPerHost

  number of processors available on each of the remote hosts

- MessageHandler * ideaMessageHandler

  data required by the IDEA framework

8.42.1 Detailed Description

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

This class is currently a placeholder.

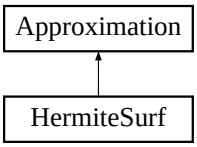
The documentation for this class was generated from the following files:

- GridApplicInterface.H
- GridApplicInterface.C

8.43 HermiteSurf Class Reference

Derived approximation class for Hermite polynomials (global approximation).

Inheritance diagram for HermiteSurf::



Public Member Functions

- HermiteSurf (const ProblemDescDB &problem_db, const size_t &num_acv)

  constructor

- ~HermiteSurf ()

  destructor

Protected Member Functions

- int required_samples ()

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

- const RealVector & approximation_coefficients ()

  return the coefficient array computed by find_coefficients()

- void find_coefficients ()

  find the Polynomial Chaos coefficients for the response surface

- 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∗numCurrentPoints entries

- int numChaos

  Number of terms in Polynomial Chaos Expansion.

- int highestOrder

  Highest order of Hermite Polynomials in Expansion.

8.43.1 Detailed Description

Derived approximation class for Hermite polynomials (global approximation).

The HermiteSurf 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).

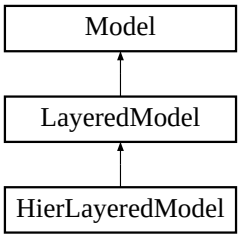
The documentation for this class was generated from the following files:

- HermiteSurf.H
- HermiteSurf.C

8.44 HierLayeredModel Class Reference

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

Inheritance diagram for HierLayeredModel::



Public Member Functions

- HierLayeredModel (ProblemDescDB &problem_db)
  constructor
- ~HierLayeredModel ()
  destructor

Protected Member Functions

- void derived_compute_response (const IntArray &asv)
  portion of compute_response() specific to HierLayeredModel
- void derived_asynch_compute_response (const IntArray &asv)
  portion of asynch_compute_response() specific to HierLayeredModel
- const ResponseArray & derived_synchronize ()
  portion of synchronize() specific to HierLayeredModel
- const ResponseList & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to HierLayeredModel
- const IntList & synchronize_nowait_completions ()
  return completion id's matching response list from derived_synchronize_nowait()
- Model subordinate_model ()
  return highFidelityModel
- Interface & interface ()
  return lowFidelityInterface

- void layering_bypass (bool bypass_flag)

  set layeringBypass flag and pass request on to highFidelityModel for any lower-level layerings.

- void build_approximation ()

  use highFidelityModel to compute the truth values needed for correction of lowFidelityInterface results

- void component_parallel_mode (int mode)

  update component parallel mode for supporting parallelism in lowFidelityInterface and highFidelityModel

- String local_eval_synchronization ()

  return lowFidelityInterface local evaluation synchronization setting

- int local_eval_concurrency ()

  return lowFidelityInterface asynchronous evaluation concurrency

- bool derived_master_overload () const

  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to low-FidelityInterface)

- void derived_init_communicators (const int &max_iterator_concurrency)

  set up lowFidelityInterface and highFidelityModel for parallel operations

- void derived_init_serial ()

  set up lowFidelityInterface and highFidelityModel for serial operations.

- void reset_communicators ()

  reset communicator partition data for the HierLayeredModel (request forwarded to lowFidelityInterface and highFidelityModel)

- void free_communicators ()

  deallocate communicator partitions for the HierLayeredModel (request forwarded to lowFidelityInterface and highFidelityModel)

- void serve ()

  Service lowFidelityInterface 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 lowFidelityInterface and highFidelityModel server operations when iteration on the HierLayeredModel is complete.

- int total_eval_counter () const

  return the total evaluation count for the HierLayeredModel (request forwarded to lowFidelityInterface)

- int new_eval_counter () const

  return the new evaluation count for the HierLayeredModel (request forwarded to lowFidelityInterface)

Private Member Functions

- void update_high_fidelity_model ()

  update highFidelityModel with current variable values/bounds/labels

Private Attributes

- Interface lowFidelityInterface

  manages the approximate low fidelity function evaluations

- Model highFidelityModel

  provides truth evaluations for computing corrections to the low fidelity results

- 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.

- IntList evalIdList

  bookkeeps fnEvalId's for correction of asynchronous low fidelity evaluations

8.44.1 Detailed Description

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

The HierLayeredModel 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 lowFidelityInterface which manages the approximate low fidelity function evaluations and a highFidelityModel which provides truth evaluations for computing corrections to the low fidelity results.

8.44.2 Member Function Documentation

8.44.2.1 void derived_compute_response (const IntArray & *asv*) [protected, virtual]

portion of compute_response() specific to HierLayeredModel

Evaluate the approximate response using lowFidelityInterface, compute the high fidelity response with build_approximation() (if not performed previously), and, if correction is active, correct the low fidelity results.

Reimplemented from Model.

8.44.2.2 void derived_asynch_compute_response (const IntArray & *asv*) [protected, virtual]

portion of asynch_compute_response() specific to HierLayeredModel

Evaluate the approximate response using an asynchronous lowFidelityInterface mapping 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.

8.44.2.3 const ResponseArray & derived_synchronize () [protected, virtual]

portion of synchronize() specific to HierLayeredModel

Perform a blocking retrieval of all asynchronous evaluations from lowFidelityInterface and, if automatic correction is on, apply correction to each response in the array.

Reimplemented from Model.

8.44.2.4 const ResponseList & derived_synchronize_nowait () [protected, virtual]

portion of synchronize_nowait() specific to HierLayeredModel

Perform a nonblocking retrieval of currently available asynchronous evaluations from lowFidelityInterface and, if automatic correction is on, apply correction to each response in the list.

Reimplemented from Model.

8.44.2.5 String local_eval_synchronization () [inline, protected, virtual]

return lowFidelityInterface local evaluation synchronization setting

Used in setting Model::asynchEvalFlag. highFidelityModel synchronization is used for setting asynchEvalFlag within highFidelityModel.

Reimplemented from Model.

8.44.2.6 int local_eval_concurrency () [inline, protected, virtual]

return lowFidelityInterface asynchronous evaluation concurrency

Used in setting Model::evaluationCapacity. highFidelityModel concurrency is used for setting evaluationCapacity within highFidelityModel.

Reimplemented from Model.

8.44.2.7 bool derived_master_overload () const [inline, protected, virtual]

flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to lowFidelityInterface)

iterator_eval_dedicated_master_flag() and multi_proc_eval_flag() flags from lowFidelityInterface are used. Derived master overload for highFidelityModel is handled separately in highFidelityModel.compute_response().

Reimplemented from Model.

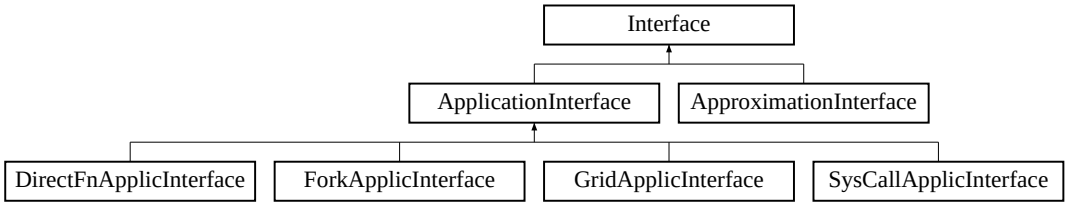
The documentation for this class was generated from the following files:

- HierLayeredModel.H
- HierLayeredModel.C

8.45 Interface Class Reference

Base class for the interface class hierarchy.

Inheritance diagram for Interface::



Public Member Functions

- Interface ()
  default constructor
- Interface (ProblemDescDB &problem_db, const size_t &num_acv, const size_t &num_fns)
  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 IntArray &asv, 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 ResponseList & 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 () const

  returns the minimum number of samples required to build a particular ApproximationInterface (used by SurrLayeredModels).

- virtual void build_global_approximation (Iterator &dace_iterator, const RealVector &lower_bnds, const RealVector &upper_bnds)

  builds a global approximation for use as a surrogate

- virtual void build_local_approximation (Model &actual_model)

  builds a local approximation for use as a surrogate

- virtual void update_approximation (const RealVector &x_star, const Response &response_star)

  updates an existing global approximation with new data

- virtual const RealVectorArray & approximation_coefficients ()

  retrieve the approximation coefficients from each Approximation within an ApproximationInterface

- void assign_rep (Interface *interface_rep)

  replaces existing letter with a new one

- const IntList & synch_nowait_completions ()

  returns id's matching response list from synch_nowait()

- const String & interface_type () const

  returns the interface type

- int total_eval_counter () const

  returns the total number of evaluations of the interface

- int new_eval_counter () const

  returns the number of new (nonduplicate) evaluations of 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)

Protected Attributes

- String interfaceType

  interface type may be (1) application: system, fork, direct, or grid; or (2) approximation: ann, rsm, mars, hermite, ksm, mpa, taylor, or hierarchical.

- int fnEvalId

  total evaluation counter

- int newFnEvalId

  new (non-duplicate) evaluation counter

- IntList beforeSynchIdList

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

- ResponseArray rawResponseArray

  The complete array of responses returned after a blocking schedule of asynchronous evaluations.

- ResponseList rawResponseList

  The partial list of responses returned after a nonblocking schedule of asynchronous evaluations.

- IntList completionList

  identifies the responses in rawResponseList for nonblocking schedules.

- 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, const size_t &num_acv, const size_t &num_fns)

  Used by the envelope to instantiate the correct letter class.

Private Attributes

- Interface * interfaceRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing interfaceRep

8.45.1 Detailed Description

Base class for the interface class hierarchy.

The Interface class hierarchy provides the part of a Model that is responsible for mapping a set of Variables into a set of Responses. The mapping is performed using either a simulation-based application interface or a surrogate-based approximation interface. For memory efficiency and enhanced polymorphism, the interface hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Interface) serves as the envelope and one of the derived classes (selected in Interface::get_interface()) serves as the letter.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 Interface ()

default constructor

used in Model envelope class instantiations

8.45.2.2 Interface (ProblemDescDB & *problem_db*, const size_t & *num_acv*, const size_t & *num_fns*)

standard constructor for envelope

Used in Model instantiation to build the envelope. This constructor only needs to extract enough data to properly execute get_interface, since Interface::Interface(BaseConstructor, problem_db) builds the actual base class data inherited by the derived interfaces.

8.45.2.3 Interface (const Interface & *interface*)

copy constructor

Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.

8.45.2.4 ~Interface () [virtual]

destructor

Destructor decrements referenceCount and only deletes interfaceRep if referenceCount is zero.

8.45.2.5 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).

8.45.3 Member Function Documentation

8.45.3.1 Interface operator= (const Interface & *interface*)

assignment operator

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

8.45.3.2 void assign_rep (Interface * *interface_rep*)

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 is modeled after get_interface() in that it does not increment the referenceCount for the new interfaceRep.

8.45.3.3 Interface * get_interface (ProblemDescDB & *problem_db*, const size_t & *num_acv*, const size_t & *num_fns*) [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, as given by the interfaceType attribute.

8.45.4 Member Data Documentation

8.45.4.1 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.

8.45.4.2 ResponseList rawResponseList [protected]

The partial list of responses returned after a nonblocking schedule of asynchronous evaluations.

The list is a partial set of completions which must be identified through the use of completionList. Postprocessing from raw to combined form (i.e., finite difference gradient merging) is not currently supported in Model::synchronize_nowait().

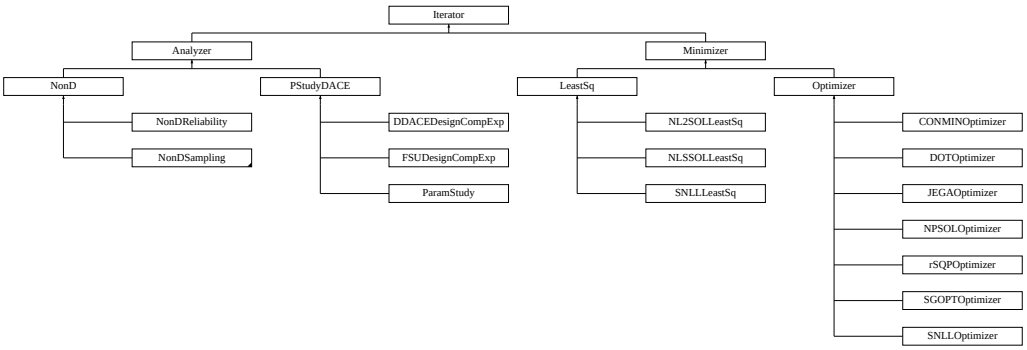
The documentation for this class was generated from the following files:

- DakotaInterface.H
- DakotaInterface.C

8.46 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_iterator ()
  run the iterator
- virtual const Variables & iterator_variable_results () const
  return the final iterator solution (variables)
- virtual const Response & iterator_response_results () const
  return the final iterator solution (response)
- virtual void print_iterator_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 assign_rep (Iterator *iterator_rep)

  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 int & maximum_concurrency () const

  return the maximum concurrency supported by the iterator

- void active_set_vector (const IntArray &asv)

  set the default active set vector (for use with iterators that employ evaluate_parameter_sets())

- void iterator_response_results_asv (const IntArray &asv)

  set the requested data for the final iterator response results

- void sub_iterator_flag (bool si_flag)

set subIteratorFlag

- 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)

Protected Attributes

- Model userDefinedModel

  shallow copy (shared rep) of the model passed into the constructor. A class member reference is not needed in this case due to the presence of representation sharing in Models.

- const ProblemDescDB & probDescDB

  class member reference to the problem description database

- String methodName

  name of the iterator (the user's method spec)

- int maxIterations

  maximum number of iterations for the iterator

- int maxFunctionEvals

  maximum number of fn evaluations for the iterator

- int numFunctions

  number of response functions

- int maxConcurrency

  maximum coarse-grained concurrency

- int numContinuousVars

  number of active continuous vars.

- int numDiscreteVars

  number of active discrete vars.

- int numVars

  total number of vars. (active and inactive)

- IntArray activeSetVector

  this vector tracks the data requirements for the response functions. It uses a 0 value for inactive functions and, for active functions, sums 1 for value, 2 for gradient, and 4 for Hessian.

- IntArray finalResultsASV

  this active set vector specifies the required final results to be returned by iterator_response_results()

- bool subIteratorFlag

  flag indicating if this Iterator is a sub-iterator (NestedModel::subIterator) or SurrLayeredModel::daceIterator).

- 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

- Iterator * iteratorRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing iteratorRep

8.46.1 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.

8.46.2 Constructor & Destructor Documentation

8.46.2.1 Iterator ()

default constructor

The default constructor is used in Vector<Iterator> instantiations and for initialization of Iterator objects contained in Strategy derived classes (see derived class header files). iteratorRep is NULL in this case (a populated problem_db is needed to build a meaningful Iterator object). This makes it necessary to check for NULL pointers in the copy constructor, assignment operator, and destructor.

8.46.2.2 Iterator (Model & *model*)

standard constructor for envelope

Used in iterator instantiations within strategy constructors. Envelope constructor only needs to extract enough data to properly execute get_iterator, since Iterator(BaseConstructor, model) builds the actual base class data inherited by the derived iterators.

8.46.2.3 Iterator (const Iterator & *iterator*)

copy constructor

Copy constructor manages sharing of iteratorRep and incrementing of referenceCount.

8.46.2.4 ~Iterator () [virtual]

destructor

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

8.46.2.5 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).

8.46.2.6 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 (e.g., ApproximationInterface instantiation of DDACEDesignCompExp or NonDSampling, NonDReliability usage of optimizers, etc.). Therefore it only sets attributes taken from the incoming model.

8.46.3 Member Function Documentation

8.46.3.1 Iterator operator= (const Iterator & *iterator*)

assignment operator

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

8.46.3.2 void run_iterator () [virtual]

run the iterator

This function is the primary run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented in LeastSq, NonD, Optimizer, and PStudyDACE.

8.46.3.3 void assign_rep (Iterator * *iterator_rep*)

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 is modeled after get_iterator() in that it does not increment the referenceCount for the new iteratorRep.

8.46.3.4 Iterator * get_iterator (Model & *model*) [private]

Used by the envelope to instantiate the correct letter class.

Used only by the envelope constructor to initialize iteratorRep to the appropriate derived type, as given by the methodName attribute.

8.46.4 Member Data Documentation

8.46.4.1 Real fdGradStepSize [protected]

relative finite difference step size for numerical gradients

A scalar value (instead of the vector fd_gradient_step_size spec) is used within the iterator hierarchy since this attribute is only used to publish a step size to vendor numerical gradient algorithms.

8.46.4.2 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.

8.46.4.3 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

8.47 JEGAEvaluator Class Reference

This evaluator uses Sandia National Laboratories Dakota software.

Public Member Functions

- const Model & GetDakotaModel () const

  Returns the "_model" object by const reference.

- virtual bool Evaluate (DesignGroup &group)

  Does evaluation of each design in "group'.

- virtual bool Evaluate (Design &des)

  This method cannot be used!!

- virtual string GetName () const

  Returns the proper name of this operator.

- virtual string GetDescription () const

  Returns a full description of what this operator does and how.

- virtual GeneticAlgorithmOperator * Clone (GeneticAlgorithm &algorithm) const

  Creates and returns a pointer to an exact duplicate of this operator.

- JEGAEvaluator (GeneticAlgorithm &alg, Model &model)

  Constructs a JEGAEvaluator for use by "alg".

- JEGAEvaluator (const JEGAEvaluator &copy)

  Copy constructs a JEGAEvaluator.

- JEGAEvaluator (const JEGAEvaluator &copy, GeneticAlgorithm &algorithm, Model &model)

  Copy constructs a JEGAEvaluator for use by "algorithm".

Static Public Member Functions

- string Name ()

  Returns the proper name of this operator.

- string Description ()

  Returns a full description of what this operator does and how.

- GeneticAlgorithmOperator * Create (GeneticAlgorithm &algorithm)

  returns a new instance of this operator class for use by "algorithm"

Protected Member Functions

- Model & GetDakotaModel ()

  Returns the "_model" object by reference.

- RealVector GetContinuumVariableValues (const Design &des) const

  Returns the continuous Design variable values held in Design "des".

- IntVector GetDiscreteVariableValues (const Design &des) const

  Returns the discrete Design variable values held in Design "des".

- void GetContinuumVariableValues (const Design &from, RealVector &into) const

  Places the continuous Design variable values from Design "from" into RealVector "into".

- void GetDiscreteVariableValues (const Design &from, IntVector &into) const

  Places the discrete Design variable values from Design "from" into IntVector "into".

- void SeparateVariables (const Design &from, IntVector &intoDisc, RealVector &intoCont) const

  This method fills "intoDisc" and "intoCont" appropriately using the values of "from".

- void RecordResponses (const RealVector &from, Design &into) const

  Records the computed objective and constraint function values into "into".

- size_t GetNumberNonLinearConstraints () const

  Returns the number of non-linear constraints for the problem.

- size_t GetNumberLinearConstraints () const

  Returns the number of linear constraints for the problem.

Private Member Functions

- JEGAEvaluator (GeneticAlgorithm &alg)

  This constructor has no implementation and cannot be used.

Private Attributes

- Model & _model

  The Model known by this evaluator.

Static Private Attributes

- const bool _is_standard_registered

  Initialization causes registry with the StandarOperatorGroup.

8.47.1 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.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 JEGAEvaluator (GeneticAlgorithm & *alg*) [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.

8.47.3 Member Function Documentation

8.47.3.1 GeneticAlgorithmOperator * Create (GeneticAlgorithm & *algorithm*) [static]

returns a new instance of this operator class for use by "algorithm"

This method cannot be used. It is provided so that this operator can still be registered in operator groups. Attempts to use this method will result in program abort.

8.47.3.2 RealVector GetContinuumVariableValues (const Design & *des*) const [protected]

Returns the continuous Design variable values held in Design "des".

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.

8.47.3.3 IntVector GetDiscreteVariableValues (const Design & *des*) const [protected]

Returns the discrete Design variable values held in Design "des".

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.

8.47.3.4 void GetContinuumVariableValues (const 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.

8.47.3.5 void GetDiscreteVariableValues (const 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.

8.47.3.6 void SeparateVariables (const Design & *from*, IntVector & *intoDisc*, RealVector & *intoCont*) const [protected]

This method fills "intoDisc" and "intoCont" appropriately using the values of "from".

It is more efficient to use this method than to use GetDiscreateVariableValues and GetContinuumVariable-Values separately if you want both.

8.47.3.7 void RecordResponses (const RealVector & *from*, Design & *into*) const [protected]

Records the computed objective and constraint function values into "into".

This method takes the response values stored in "from" and properly transfers them into the "into" design.

8.47.3.8 bool Evaluate (DesignGroup & *group*) [virtual]

Does evaluation of each design in "group'.

This method uses the Model know 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.

8.47.3.9 bool Evaluate (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.

8.47.4 Member Data Documentation

8.47.4.1 const bool _is_standard_registered [static, private]

Initial value:

```
StandardOperatorGroup::EvaluatorRegistry().Register(
     JEGAEvaluator::Name(), &JEGAEvaluator::Create)
```

Initialization causes registry with the StandarOperatorGroup.

This flag indicates whether or not this class was properly registered with the StandardOperatorGroup on startup. The JEGAEvaluator is a special case that registers itself with the group instead of having the group register it.

8.47.4.2 Model& _model [private]

The Model known by this evaluator.

It is through this model that evaluations will take place.

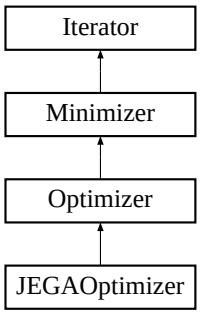
The documentation for this class was generated from the following files:

- JEGAEvaluator.H
- JEGAEvaluator.C

8.48 JEGAOptimizer Class Reference

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

Inheritance diagram for JEGAOptimizer::



Public Member Functions

- const GeneticAlgorithm & GetTheGA () const
  Returns the JEGA being used to optimize the problem (const).
- GeneticAlgorithm & GetTheGA ()
  Returns the JEGA being used to optimize the problem (non-const).
- const DesignTarget & GetTheTarget () const
  Returns the DesignTarget created here being used by the GA (const).
- DesignTarget & GetTheTarget ()
  Returns the DesignTarget created here being used by the GA (non-const).
- virtual void find_optimum ()
  Performs the iterations to determine the optimal set of solution.
- JEGAOptimizer (Model &model, const string &method)
  Constructs a JEGAOptimizer class object.
- ~JEGAOptimizer ()
  Destructs a JEGAOptimizer.

Protected Member Functions

- void CreateTheGA ()
  This method creates the GA.
- void LoadTheGA ()

Loads required information into a GA.

- void CreateTheTarget ()

  This method creates but doesn't load the DesignTarget.

- void LoadTheTarget ()

  This method creates but doesn't load the DesignTarget.

- void CreateDesignVariableInfos ()

  Creates but doesn't load DesignVariableInfo objects.

- void LoadDesignVariableInfos ()

  Loads information into the DesignVariableInfo objects.

- void CreateConstraintInfos ()

  Creates but doesn't load ConstraintInfo objects.

- void LoadConstraintInfos ()

  Loads information into the ConstraintInfo objects.

- void ExtractOperatorParameters (GeneticAlgorithmOperator *op)

  This method requests that "op" retrieve its parameter values from "params".

- void VerifyValidOperator (GeneticAlgorithmOperator *op, const string &str)

  This method verifies that "op" is not null.

Private Attributes

- GeneticAlgorithm * _theGA

  This is a pointer to the instantiated GeneticAlgorithm.

- DesignTarget * _theTarget

  This is a pointer to the DesignTarget object for the GeneticAlgorithm.

- JEGAEvaluator * _theEvaluator

  A persistant pointer to the Evaluator created for the GeneticAlgorithm.

- string _method

  The type of GA to create. Currently one of "moga" and "soga".

Static Private Attributes

- string _sogaMethodText

  The text that indicates the SOGA method.

- string _mogaMethodText

  The text that indicates the MOGA method.

8.48.1 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 Genetic-Algorithm.

8.48.2 Constructor & Destructor Documentation

8.48.2.1 JEGAOptimizer (Model & *model*, const string & *method*)

Constructs a JEGAOptimizer class object.

This method does much of the initialization work for the algorithm.

8.48.3 Member Function Documentation

8.48.3.1 void CreateTheGA () [protected]

This method creates the GA.

It instantiates the GA and all the operators.

8.48.3.2 void LoadTheGA () [protected]

Loads required information into a GA.

This method must be called prior to attempting any optimization with the GA. It does what is necessary to load the target properly.

8.48.3.3 void CreateTheTarget () [protected]

This method creates but doesn't load the DesignTarget.

It instantiates the Target and the associated information objects. The information however is not considered current until LoadTheTarget is called (which should not be done in the constructor).

8.48.3.4 void LoadTheTarget () [protected]

This method creates but doesn't load the DesignTarget.

This method must be called prior to attempting any optimization with the GA. It does what is necessary to load the target properly.

8.48.3.5 void CreateDesignVariableInfos () [protected]

Creates but doesn't load DesignVariableInfo objects.

This method records the info objects with the target which must already have been created.

8.48.3.6 void LoadDesignVariableInfos () [protected]

Loads information into the DesignVariableInfo objects.

Information includes stuff like bounds, labels, discrete values, etc.

8.48.3.7 void CreateConstraintInfos () [protected]

Creates but doesn't load ConstraintInfo objects.

This method records the info objects with the target which must already have been created.

8.48.3.8 void LoadConstraintInfos () [protected]

Loads information into the ConstraintInfo objects.

Information includes stuff like targets and bounds, labels, and coefficients for linear constraints.

8.48.3.9 void ExtractOperatorParameters (GeneticAlgorithmOperator * *op*) [protected]

This method requests that "op" retrieve its parameter values from "params".

If "op" is unable to do so, this method causes an abort.

8.48.3.10 void VerifyValidOperator (GeneticAlgorithmOperator * *op*, const string & *str*) [protected]

This method verifies that "op" is not null.

If it is, this method causes an abort.

8.48.3.11 void find_optimum () [virtual]

Performs the iterations to determine the optimal set of solution.

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

8.49 KrigApprox Class Reference

Utility class for kriging interpolation.

Public Member Functions

- KrigApprox (int, int, const RealVector &, const RealVector &, const RealVector &)
  constructor
- ~KrigApprox ()
  destructor
- void ModelBuild (int, int, const RealVector &, const RealVector &, bool)
  Function to compute vector and matrix terms in the kriging surface.
- Real ModelApply (int, int, const RealVector &)
  Function returns a response value using the kriging surface.

Private Attributes

- 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 conminSingleArray
  Array size parameter needed in interface to CONMIN.
- int numcon
  CONMIN variable: Number of constraints.
- int NFDG
  CONMIN variable: Finite difference flag.
- int IPRINT

CONMIN variable: Flag to control amount of output data.

- int ITMAX

  CONMIN variable: Flag to specify the maximum number of iterations.

- Real FDCH

  CONMIN variable: Relative finite difference step size.

- Real FDCHM

  CONMIN variable: Absolute finite difference step size.

- Real CT

  CONMIN variable: Constraint thickness parameter.

- Real CTMIN

  CONMIN variable: Minimum absolute value of CT used during optimization.

- Real CTL

  CONMIN variable: Constraint thickness parameter for linear and side constraints.

- Real CTLMIN

  CONMIN variable: Minimum value of CTL used during optimization.

- Real DELFUN

  CONMIN variable: Relative convergence criterion threshold.

- Real DABFUN

  CONMIN variable: Absolute convergence criterion threshold.

- int conminInfo

  CONMIN variable: status flag for optimization.

- Real * S

  Internal CONMIN array.

- Real * G1

  Internal CONMIN array.

- Real * G2

  Internal CONMIN array.

- Real * B

  Internal CONMIN array.

- Real * C

  Internal CONMIN array.

- int * MS1

  Internal CONMIN array.

- Real * SCAL

  Internal CONMIN array.

- Real * DF

  Internal CONMIN array.

- Real * A

  Internal CONMIN array.

- int * ISC

  Internal CONMIN array.

- int * IC

  Internal CONMIN array.

- Real * conminThetaVars

  Temporary array of design variables used by CONMIN (length N1 = numdv+2).

- Real * conminThetaLowerBnds

  Temporary array of lower bounds used by CONMIN (length N1 = numdv+2).

- Real * conminThetaUpperBnds

  Temporary array of upper bounds used by CONMIN (length N1 = numdv+2).

- Real ALPHAX

  Internal CONMIN variable: 1-D search parameter.

- Real ABOBJ1

  Internal CONMIN variable: 1-D search parameter.

- Real THETA

  Internal CONMIN variable: mean value of push-off factor.

- Real PHI

  Internal CONMIN variable: "participation coefficient".

- int NSIDE

  Internal CONMIN variable: side constraints parameter.

- int NSCAL

  Internal CONMIN variable: scaling control parameter.

- int NACMX1

  Internal CONMIN variable: estimate of 1+(max # of active constraints).

- int LINOBJ

  Internal CONMIN variable: linear objective function identifier (unused).

- int ITRM

  Internal CONMIN variable: diminishing return criterion iteration number.

- int ICNDIR

  Internal CONMIN variable: conjugate direction restart parameter.

- int IGOTO

  Internal CONMIN variable: internal optimization termination flag.

- int NAC

  Internal CONMIN variable: number of active and violated constraints.

- int INFOG

  Internal CONMIN variable: gradient information flag.

- int ITER

  Internal CONMIN variable: iteration count.

- int iFlag

  Fortran77 flag for kriging computations.

- Real betaHat

  Estimate of the beta term in the kriging model..

- Real maxLikelihoodEst

  Error term computed via Maximum Likelihood Estimation.

- int numNewPts

  Size variable for the arrays used in kriging computations.

- int numSampQuad

  Size variable for the arrays used in kriging computations.

- Real * thetaVector

  Array of correlation parameters for the kriging model.

- Real * xMatrix

  A 2-D array of design points used to build the kriging model.

- Real * yValueVector

  Array of response values corresponding to the array of design points.

- Real * xNewVector

  A 2-D array of design points where the kriging model will be evaluated.

- Real * yNewVector

  Array of response values corresponding to the design points specified in xNewVector.

- Real * thetaLoBndVector

  Array of lower bounds in optimizer-to-kriging interface.

- Real * thetaUpBndVector

Array of upper bounds in optimizer-to-kriging interface.

- Real * constraintVector

  Array of constraint values (used with optimizer).

- Real * rhsTermsVector

  Internal array for kriging Fortran77 code: matrix algebra result.

- int * iPivotVector

  Internal array for kriging Fortran77 code: pivot vector for linear algebra.

- Real * correlationMatrix

  Internal array for kriging Fortran77 code: correlation matrix.

- Real * invcorrelMatrix

  Internal array for kriging Fortran77 code: inverse correlation matrix.

- Real * fValueVector

  Internal array for kriging Fortran77 code: response value vector.

- Real * fRinvVector

  *Internal array for kriging Fortran77 code: vector*matrix result.*

- Real * yfbVector

  Internal array for kriging Fortran77 code: vector arithmetic result.

- Real * yfbRinvVector

  *Internal array for kriging Fortran77 code: vector*matrix result.*

- Real * rXhatVector

  Internal array for kriging Fortran77 code: local correlation vector.

- Real * workVector

  Internal array for kriging Fortran77 code: temporary storage.

- Real * workVectorQuad

  Internal array for kriging Fortran77 code: temporary storage.

- int * iworkVector

  Internal array for kriging Fortran77 code: temporary storage.

8.49.1 Detailed Description

Utility class for kriging interpolation.

The KrigApprox class provides utilities for the KrigingSurf class. It is based on the Ph.D. thesis work of Tony Giunta.

8.49.2 Member Function Documentation

8.49.2.1 Real ModelApply (int, int, const RealVector &)

Function returns a response value using the kriging surface.

The response value is computed at the design point specified by the RealVector function argument.

8.49.3 Member Data Documentation

8.49.3.1 int N1 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N1 = number of variables + 2

8.49.3.2 int N2 [private]

Size variable for CONMIN arrays. See CONMIN manual.

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

8.49.3.3 int N3 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N3 = Maximum possible number of active constraints.

8.49.3.4 int N4 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N4 = Maximum(N3,number of variables)

8.49.3.5 int N5 [private]

Size variable for CONMIN arrays. See CONMIN manual.

N5 = 2∗(N4)

8.49.3.6 Real CT [private]

CONMIN variable: Constraint thickness parameter.

The value of CT decreases in magnitude during optimization.

8.49.3.7 Real* S [private]

Internal CONMIN array.

Move direction in N-dimensional space.

8.49.3.8 Real* G1 [private]

Internal CONMIN array.

Temporary storage of constraint values.

8.49.3.9 Real* G2 [private]

Internal CONMIN array.

Temporary storage of constraint values.

8.49.3.10 Real* B [private]

Internal CONMIN array.

Temporary storage for computations involving array S.

8.49.3.11 Real* C [private]

Internal CONMIN array.

Temporary storage for use with arrays B and S.

8.49.3.12 int* MS1 [private]

Internal CONMIN array.

Temporary storage for use with arrays B and S.

8.49.3.13 Real* SCAL [private]

Internal CONMIN array.

Vector of scaling parameters for design parameter values.

8.49.3.14 Real* DF [private]

Internal CONMIN array.

Temporary storage for analytic gradient data.

8.49.3.15 Real* A [private]

Internal CONMIN array.

Temporary 2-D array for storage of constraint gradients.

8.49.3.16 int* ISC [private]

Internal CONMIN array.

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

8.49.3.17 int* IC [private]

Internal CONMIN array.

Array of flags to identify active and violated constraints

8.49.3.18 int iFlag [private]

Fortran77 flag for kriging computations.

iFlag=1 computes vector and matrix terms for the kriging surface, iFlag=2 computes the response value (using kriging) at the user-supplied design point.

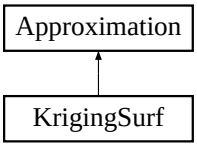
The documentation for this class was generated from the following files:

- KSMSurf.H
- KSMSurf.C

8.50 KrigingSurf Class Reference

Derived approximation class for kriging interpolation.

Inheritance diagram for KrigingSurf::



Public Member Functions

- KrigingSurf (const ProblemDescDB &problem_db, const size_t &num_acv)
  constructor
- ~KrigingSurf ()
  destructor

Protected Member Functions

- void find_coefficients ()
  calculate the data fit coefficients using the currentPoints list of SurrogateDataPoints
- int required_samples ()
  return the minimum number of samples required to build the derived class approximation type in numVars dimensions
- Real get_value (const RealVector &x)
  retrieve the approximate function value for a given parameter vector

Private Attributes

- KrigApprox * krigObject
  Kriging Surface object declaration.
- RealVector x_matrix
  A 2-d array of all sample sites (design points) used to create the kriging surface.
- RealVector f_of_x_array
  An array of response values; one response value per sample site.
- RealVector correlationVector

An array of correlation parameter values used to build the kriging surface.

- bool runConminFlag

  Flag to run CONMIN (value=1) or use user-supplied correlations (value=0).

8.50.1 Detailed Description

Derived approximation class for kriging interpolation.

The KrigingSurf class uses a the kriging approach to interpolate between data points. It is based on the Ph.D. thesis work of Tony Giunta.

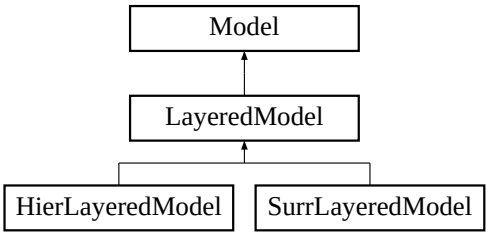
The documentation for this class was generated from the following files:

- KSMSurf.H
- KSMSurf.C

8.51 LayeredModel Class Reference

Base class for the layered models (SurrLayeredModel and HierLayeredModel).

Inheritance diagram for LayeredModel::



Protected Member Functions

- LayeredModel (ProblemDescDB &problem_db)

  constructor

- ~LayeredModel ()

  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 LayeredModel attributes and attributes of the submodel (SurrLayeredModel::actualModel or HierLayeredModel::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

- ResponseArray correctedResponseArray

  array of corrected responses used in derived_synchronize() functions

- ResponseList correctedResponseList

  list of corrected responses used in derived_synchronize_nowait() functions

- RealVectorList rawCVarsList

  list 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

- size_t approxBuilds

  number of calls to build_approximation()

- bool autoCorrection

  a flag which controls the use of apply_correction() in SurrLayeredModel and HierLayeredModel approximate response computations

- bool layeringBypass

  a flag which allows bypassing the approximation for evaluations on the underlying truth model.

- String approxType

  approximation type identifier string: global, local, or hierarchical

- String refitInactive

  flag denoting a user setting for rebuilding the approximation when changes occur to the inactive variables data.

- RealVector fitInactiveCVars

  stores a copy of the inactive continuous variables when the approximation is built; used to detect when a rebuild is required.

- RealVector fitInactiveCLowerBnds

  stores a copy of the inactive continuous lower bounds when the approximation is built; used to detect when a rebuild is required.

- RealVector fitInactiveCUpperBnds

  stores a copy of the inactive continuous upper bounds when the approximation is built; used to detect when a rebuild is required.

- IntVector fitInactiveDVars

  stores a copy of the inactive discrete variables when the approximation is built; used to detect when a rebuild is required.

- IntVector fitInactiveDLowerBnds

  stores a copy of the inactive discrete lower bounds when the approximation is built; used to detect when a rebuild is required.

- IntVector fitInactiveDUpperBnds

  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 IntArray &asv)

  internal convenience function for applying additive corrections

- void apply_multiplicative_correction (RealVector &beta_corrected_fns, RealMatrix &beta_corrected_grads, RealMatrixArray &beta_corrected_hessians, const String &approx_interf_id, const RealVector &c_vars, const IntArray &asv)

  internal convenience function for applying multiplicative corrections

Private Attributes

- bool correctionComputed

  flag indicating whether or not a correction is available

- 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

8.51.1 Detailed Description

Base class for the layered models (SurrLayeredModel and HierLayeredModel).

The LayeredModel class provides common functions to derived classes for computing and applying corrections to approximations.

8.51.2 Member Function Documentation

8.51.2.1 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.

8.51.2.2 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 refitInactive setting, and whether any inactive data has changed since the last build.

8.51.3 Member Data Documentation

8.51.3.1 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().

8.51.3.2 bool autoCorrection [protected]

a flag which controls the use of apply_correction() in SurrLayeredModel and HierLayeredModel approximate response computations

the default is on (true) once compute_correction() has been called. However this should be overridden when a new correction is desired, since compute_correction() no longer automatically backs out an old correction.

8.51.3.3 String refitInactive [protected]

flag denoting a user setting for rebuilding the approximation when changes occur to the inactive variables data.

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).

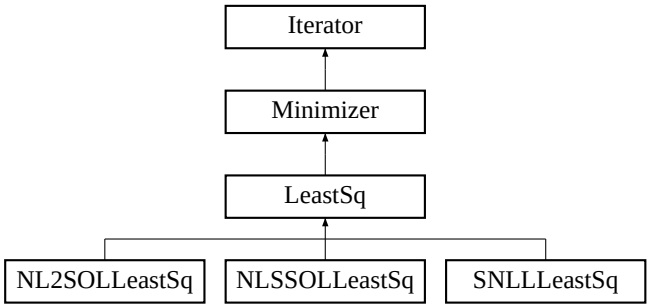
The documentation for this class was generated from the following files:

- LayeredModel.H
- LayeredModel.C

8.52 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_iterator ()
  run the iterator
- void print_iterator_results (ostream &s) const
- 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

8.52.1 Detailed Description

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

The LeastSq class provides common data and functionality for NLSSOLLeastSq and SNLLLeastSq.

8.52.2 Constructor & Destructor Documentation

8.52.2.1 LeastSq (Model & *model*) [protected]

standard constructor

This constructor extracts the inherited data for the least squares branch and performs sanity checking on gradient and constraint settings.

8.52.3 Member Function Documentation

8.52.3.1 void run_iterator () [inline, protected, virtual]

run the iterator

This function is the primary run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

8.52.3.2 void print_iterator_results (ostream & *s*) const [protected, virtual]

Redefines default iterator results printing to include optimization results (objective function and constraints).

Reimplemented from Iterator.

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

- DakotaLeastSq.H
- DakotaLeastSq.C

8.53 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 print (ostream &s) const
  Prints a List to an output stream.
- void read (MPIUnpackBuffer &s)
  Reads a List from an MPIUnpackBuffer after an MPI receive.
- void print (MPIPackBuffer &s) const
  Prints 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(*testFun)(const T &, void *), void *d, T &k) const
  Returns TRUE if the list contains an object which the user defined function finds and sets k to this object.
- size_t index (bool(*testFun)(const T &, void *), void *d) const
  Returns the index of object which the user defined test function finds.
- void sort (bool(*sortFun)(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).

8.53.1 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 DakotaVal-List class

8.53.2 Member Function Documentation

8.53.2.1 T get ()

Removes and returns the first item in the list.

Remove and return item from front of list. Returns the object pointed to by the list::begin() iterator. It also deletes the first node by calling the list::pop_front() method. Note: get() is not the same as list::front() since the latter would return the 1st item but would not delete it.

8.53.2.2 T removeAt (size_t *index*)

Removes and returns the item at the specified index.

Removes the item at the index specified. Uses the STL advance() function to step to the appropriate position in the list and then calls the list::erase() method.

8.53.2.3 bool remove (const T & *a*)

Removes the specified item from the list.

Removes the first instance matching object a from the list (and therefore differs from the STL list::remove() which removes all instances). Uses the STL find() algorithm to find the object and the list::erase() method to perform the remove.

8.53.2.4 void insert (const T & *a*) [inline]

Adds the item a to the end of the list.

Insert item at the end of list, calls list::push_back() method which places the object at the end of the list.

8.53.2.5 bool contains (const T & *a*) const [inline]

Returns TRUE if list contains object a, returns FALSE otherwise.

Uses the STL find() algorithm to locate the first instance of object a. Returns true if an instance is found.

8.53.2.6 bool find (bool(* *testFun*)(const T &, void *), void * *d*, T & *k*) const

Returns TRUE if the list contains an object which 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.

8.53.2.7 size_t index (bool(* *testFun*)(const T &, void *), void * *d*) const

Returns the index of object which the user defined test function finds.

Returns the index of the first item in the list which satisfies the test function. Uses a single list traversal to both locate the object and return its index (generic algorithms would require two loop traversals).

8.53.2.8 void sort (bool(* *sortFun*)(const T &, const T &)) [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 sortFun. 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.

8.53.2.9 size_t index (const T & *a*) const

Returns the index of the object.

Returns the index of the first item in the list which matches the object a. Uses a single list traversal to both locate the object and return its index (generic algorithms would require two loop traversals).

8.53.2.10 size_t count (const T & *a*) const [inline]

Returns the number of items in the list equal to object.

Uses the STL count() algorithm to return the number of occurences of the specified object.

8.53.2.11]

T & operator[ ] (size_t *i*)

Returns the object at index i (can use as lvalue).

Returns item at position i of the list by stepping through the list using forward or reverse STL iterators (depending on which end of the list is closer to the desired item). Once the object is found, it returns the value pointed to by the iterator.

This functionality is inefficient in 0->len loop-based list traversals and is being replaced by iterator-based list traversals in the main DAKOTA code. For isolated look-ups of a particular index, however, this approach is acceptable.

8.53.2.12]

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

Returns the object at index i, const (can't use as lvalue).

Returns const item at position i of the list by stepping through the list using forward or reverse STL iterators (depending on which end of the list is closer to the desired item). Once the object is found it returns the value pointed to by the iterator.

This functionality is inefficient in 0->len loop-based list traversals and is being replaced by iterator-based list traversals in the main DAKOTA code. For isolated look-ups of a particular index, however, this approach is acceptable.

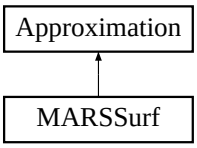
The documentation for this class was generated from the following file:

- DakotaList.H

8.54 MARSSurf Class Reference

Derived approximation class for multivariate adaptive regression splines.

Inheritance diagram for MARSSurf::



Public Member Functions

- MARSSurf (const ProblemDescDB &problem_db, const size_t &num_acv)

  constructor

- ~MARSSurf ()

  destructor

Protected Member Functions

- int required_samples ()

  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 the currentPoints list of SurrogateDataPoints

- Real get_value (const RealVector &x)

  retrieve the approximate function value for a given parameter vector

Private Attributes

- int * flags

  variable type declarations (ordinal, excluded, categorical)

- Mars * marsObject

  pointer to the Mars object (MARS wrapper provided as part of DDACE)

8.54.1 Detailed Description

Derived approximation class for multivariate adaptive regression splines.

The MARSSurf class provides a global approximation based on regression splines. It employs the C++ wrapper developed by the DDACE team for the Multivariate Adaptive Regression Splines (MARS) package from Prof. Jerome Friedman of Stanford University Dept. of Statistics.

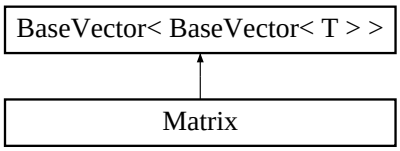
The documentation for this class was generated from the following files:

- MARSSurf.H
- MARSSurf.C

8.55 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 print (ostream &s, bool rtn) const

  Prints a Matrix to an output stream.

- void print_row_vector (ostream &s, size_t i, bool rtn) const

  Prints a Matrix to an output stream.

- void read (MPIUnpackBuffer &s)

  Reads a Matrix from an MPIUnpackBuffer after an MPI receive.

- void print (MPIPackBuffer &s) const

  Prints a Matrix to a MPIPackBuffer prior to an MPI send.

8.55.1 Detailed Description

template<class T> class Dakota::Matrix< T >

Template class for the Dakota numerical matrix.

A matrix class template to provide 2D arrays of objects. The matrix is zero-based, rows: 0 to (numRows-1) and cols: 0 to (numColumns-1). The class supports overloading of the subscript operator allowing it to emulate a normal built-in 2D array type. Matrix relies on the BaseVector template class to manage any differences between underlying DAKOTA_BASE_VECTOR implementations (RW, STL, etc.).

8.55.2 Member Function Documentation

8.55.2.1 Matrix< T > & operator= (const T & *val*) [inline]

Sets all elements in the matrix to ival.

calls base class operator=(ival)

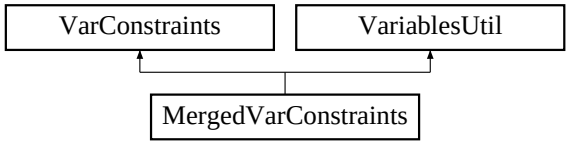
The documentation for this class was generated from the following file:

- DakotaMatrix.H

8.56 MergedVarConstraints Class Reference

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

Inheritance diagram for MergedVarConstraints::



Public Member Functions

- MergedVarConstraints (const ProblemDescDB &problem_db)
  constructor
- ~MergedVarConstraints ()
  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
- 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 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 uncertainDistLowerBnds
  the uncertain distribution lower bounds array (no discrete uncertain to merge)
- RealVector uncertainDistUpperBnds
  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)

8.56.1 Detailed Description

Derived class within the VarConstraints 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 MergedVarConstraints 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 (uncertainDistLowerBnds, uncertainDistUpperBnds), and merged state bounds arrays (mergedStateLowerBnds, mergedStateUpperBnds). The branch and bound strategy uses this approach (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to the VarConstraints constructor in Model).

8.56.2 Constructor & Destructor Documentation

8.56.2.1 MergedVarConstraints (const ProblemDescDB & *problem_db*)

constructor

Extract fundamental lower and upper bounds and merge continuous and discrete domains to create mergedDesignLowerBnds, mergedDesignUpperBnds, mergedStateLowerBnds, and mergedStateUpperBnds using utilities from VariablesUtil (uncertain distribution bounds do not require any merging).

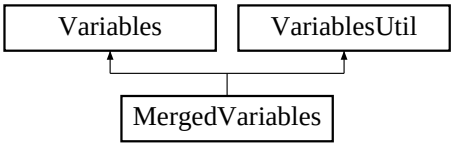
The documentation for this class was generated from the following files:

- MergedVarConstraints.H
- MergedVarConstraints.C

8.57 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)
  standard constructor
- ~MergedVariables ()
  destructor
- size_t tv () const
  Returns total number of vars.
- size_t cv () const
  Returns number of active continuous vars.
- size_t dv () const
  Returns number of active discrete 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
- 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
- size_t adv () const
  returns total number of discrete vars
- RealVector all_continuous_variables () const
  returns a single array with all continuous variables
- IntVector all_discrete_variables () const
  returns a single array with all discrete variables
- 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 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

8.57.1 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)).

8.57.2 Constructor & Destructor Documentation

8.57.2.1 MergedVariables (const ProblemDescDB & *problem_db*)

standard constructor

Extract fundamental variable types and labels and merge continuous and discrete domains to create mergedDesignVars, mergedStateVars, mergedDesignLabels, and mergedStateLabels using utilities from VariablesUtil (uncertain variables and labels do not require any merging).

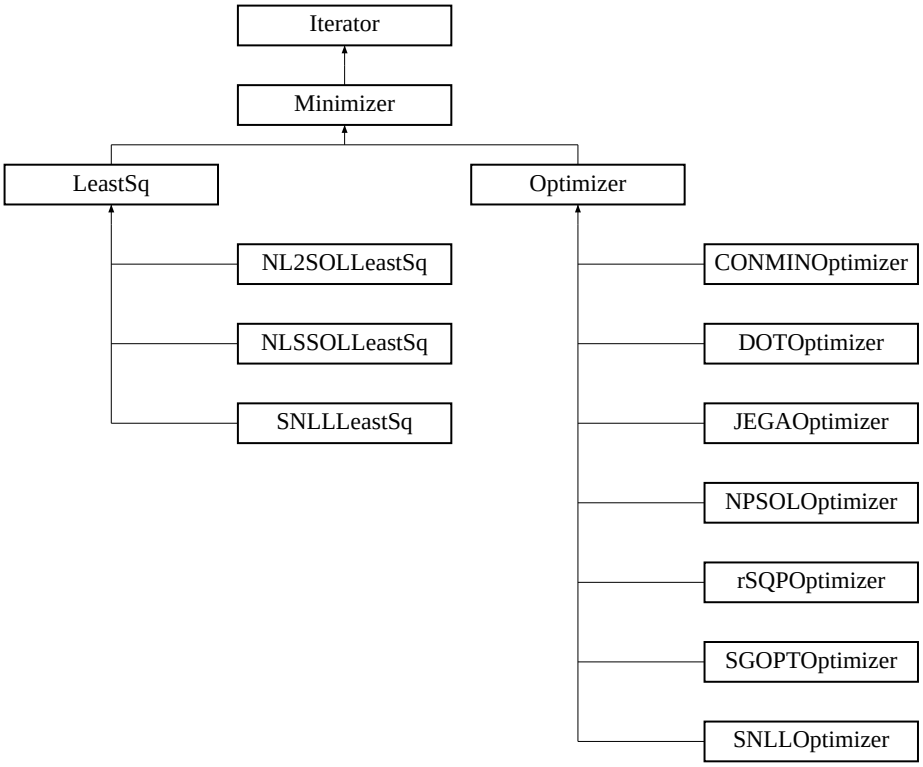
The documentation for this class was generated from the following files:

- MergedVariables.H
- MergedVariables.C

8.58 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 & iterator_variable_results () const
  return the final iterator solution (variables)
- const Response & iterator_response_results () const
  return the final iterator solution (response)

Protected Member Functions

- Minimizer ()
  default constructor
- Minimizer (Model &model)
  standard constructor

- ~Minimizer ()
  destructor

Protected Attributes

- Real convergenceTol
  optimizer/least squares convergence tolerance
- Real constraintTol
  optimizer/least squares constraint tolerance
- size_t numNonlinearIneqConstraints
  number of nonlinear inequality constraints
- RealVector nonlinearIneqLowerBnds
  nonlinear inequality constraint lower bounds
- RealVector nonlinearIneqUpperBnds
  nonlinear inequality constraint upper bounds
- Real bigRealBoundSize
  cutoff value for inequality constraint and continuous variable bounds
- int bigIntBoundSize
  cutoff value for discrete variable bounds
- size_t numNonlinearEqConstraints
  number of nonlinear equality constraints
- RealVector nonlinearEqTargets
  nonlinear equality constraint targets
- size_t numLinearIneqConstraints
  number of linear inequality constraints
- RealMatrix linearIneqConstraintCoeffs
  linear inequality constraint coefficients
- RealVector linearIneqLowerBnds
  linear inequality constraint lower bounds
- RealVector linearIneqUpperBnds
  linear inequality constraint upper bounds
- size_t numLinearEqConstraints
  number of linear equality constraints

- RealMatrix linearEqConstraintCoeffs

  linear equality constraint coefficients

- RealVector linearEqTargets

  linear equality constraint targets

- 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 vendorNumericalGradFlag

  convenience flag for gradType == numerical && methodSource == vendor

- Variables bestVariables

  best variables found in solution

- Response bestResponses

  best responses found in solution

8.58.1 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.

8.58.2 Constructor & Destructor Documentation

8.58.2.1 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.

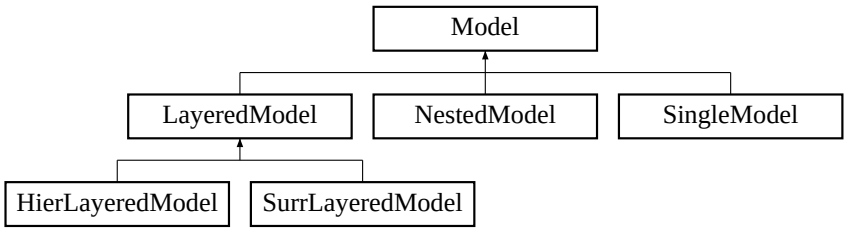
The documentation for this class was generated from the following files:

- DakotaMinimizer.H
- DakotaMinimizer.C

8.59 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 Model subordinate_model ()
  return the sub-model in nested and layered models
- virtual Iterator subordinate_iterator ()
  return the sub-iterator in nested and layered models
- virtual Interface & interface ()
  return the single interface employed by each derived model class: SingleModel::userDefinedInterface, SurrLayeredModel::approxInterface, HierLayeredModel::lowFidelityInterface, or NestedModel::optionalInterface
- virtual void layering_bypass (bool bypass_flag)
  deactivate/reactivate the approximations for any/all layered models contained within this model
- virtual void build_approximation ()
  build the approximation in LayeredModels

- virtual void update_approximation (const RealVector &x_star, const Response &response_star)

  update the approximation in SurrLayeredModels with new data

- virtual const RealVectorArray & approximation_coefficients ()

  retrieve the approximation coefficients from each Approximation within a SurrLayeredModel

- virtual void compute_correction (const Response &truth_response, const Response &approx_response, const RealVector &c_vars)

  compute correction factors for use in LayeredModels

- virtual void auto_correction (bool correction_flag)

  manages automatic application of correction factors in LayeredModels

- 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 LayeredModels)

- 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 = INTERFACE, SUBMODEL, or 0).

- 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 free_communicators ()

  deallocate communicator partitions 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 const IntList & synchronize_nowait_completions ()

  Return completion id's matching response list from synchronize_nowait.

- 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 total_eval_counter () const

  Return the total evaluation count from the interface.

- virtual int new_eval_counter () const

  Return the new (non-duplicate) evaluation count from the interface.

- void compute_response ()

  Compute the Response at currentVariables (default asv).

- void compute_response (const IntArray &asv)

  Compute the Response at currentVariables (specified asv).

- void asynch_compute_response ()

  Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (default asv).

- void asynch_compute_response (const IntArray &asv)

  Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (specified asv).

- const ResponseArray & synchronize ()

  Execute a blocking scheduling algorithm to collect the complete set of results from a group of asynchronous evaluations.

- const ResponseList & 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

- void init_serial ()

  for cases where init_communicators() will not be called, modify some default settings to behave properly in serial.

- void estimate_message_lengths ()

  estimate messageLengths for a model

- size_t tv () const

  return total number of vars

- size_t cv () const

  return number of active continuous variables

- size_t dv () const

  return number of active discrete variables

- size_t num_functions () const
  return number of functions in currentResponse
- void active_variables (const Variables &vars)
  set the active variables in currentVariables
- const RealVector & continuous_variables () const
  return the active continuous variables from currentVariables
- void continuous_variables (const RealVector &c_vars)
  set the active continuous variables in currentVariables
- const IntVector & discrete_variables () const
  return the active discrete variables from currentVariables
- void discrete_variables (const IntVector &d_vars)
  set the active discrete variables in currentVariables
- const RealVector & inactive_continuous_variables () const
  return the inactive continuous variables in currentVariables
- void inactive_continuous_variables (const RealVector &i_c_vars)
  set the inactive continuous variables in currentVariables
- const IntVector & inactive_discrete_variables () const
  return the inactive discrete variables in currentVariables
- void inactive_discrete_variables (const IntVector &i_d_vars)
  set the inactive discrete variables in currentVariables
- 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_dist_lower_bounds () const
  return the normal uncertain variable distribution lower bounds
- void normal_dist_lower_bounds (const RealVector &n_dist_lower_bnds)
  set the normal uncertain variable distribution lower bounds
- const RealVector & normal_dist_upper_bounds () const
  return the normal uncertain variable distribution upper bounds

- void normal_dist_upper_bounds (const RealVector &n_dist_upper_bnds)
  set the normal uncertain variable distribution 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_dist_lower_bounds () const
  return the lognormal uncertain variable distribution lower bounds
- void lognormal_dist_lower_bounds (const RealVector &ln_dist_lower_bnds)
  set the lognormal uncertain variable distribution lower bounds
- const RealVector & lognormal_dist_upper_bounds () const
  return the lognormal uncertain variable distribution upper bounds
- void lognormal_dist_upper_bounds (const RealVector &ln_dist_upper_bnds)
  set the lognormal uncertain variable distribution upper bounds
- const RealVector & uniform_dist_lower_bounds () const
  return the uniform uncertain variable distribution lower bounds
- void uniform_dist_lower_bounds (const RealVector &u_dist_lower_bnds)
  set the uniform uncertain variable distribution lower bounds
- const RealVector & uniform_dist_upper_bounds () const
  return the uniform uncertain variable distribution upper bounds
- void uniform_dist_upper_bounds (const RealVector &u_dist_upper_bnds)
  set the uniform uncertain variable distribution upper bounds
- const RealVector & loguniform_dist_lower_bounds () const
  return the loguniform uncertain variable distribution lower bounds
- void loguniform_dist_lower_bounds (const RealVector &lu_dist_lower_bnds)

set the loguniform uncertain variable distribution lower bounds

- const RealVector & loguniform_dist_upper_bounds () const
  return the loguniform uncertain variable distribution upper bounds
- void loguniform_dist_upper_bounds (const RealVector &lu_dist_upper_bnds)
  set the loguniform uncertain variable distribution upper bounds
- const RealVector & weibull_alphas () const
  return the weibull uncertain variable alpha parameters
- void weibull_alphas (const RealVector &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 &betas)
  set the weibull uncertain variable beta parameters
- const RealVector & weibull_dist_lower_bounds () const
  return the weibull uncertain variable distribution lower bounds
- void weibull_dist_lower_bounds (const RealVector &w_dist_lower_bnds)
  set the weibull uncertain variable distribution lower bounds
- const RealVector & weibull_dist_upper_bounds () const
  return the weibull uncertain variable distribution upper bounds
- void weibull_dist_upper_bounds (const RealVector &w_dist_upper_bnds)
  set the weibull uncertain variable distribution upper bounds
- 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 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 userDefinedVarConstraints
- void continuous_lower_bounds (const RealVector &c_l_bnds)
  set the active continuous variable lower bounds in userDefinedVarConstraints
- const RealVector & continuous_upper_bounds () const
  return the active continuous variable upper bounds from userDefinedVarConstraints
- void continuous_upper_bounds (const RealVector &c_u_bnds)
  set the active continuous variable upper bounds in userDefinedVarConstraints
- const IntVector & discrete_lower_bounds () const
  return the active discrete variable lower bounds from userDefinedVarConstraints
- void discrete_lower_bounds (const IntVector &d_l_bnds)
  set the active discrete variable lower bounds in userDefinedVarConstraints
- const IntVector & discrete_upper_bounds () const
  return the active discrete variable upper bounds from userDefinedVarConstraints
- void discrete_upper_bounds (const IntVector &d_u_bnds)
  set the active discrete variable upper bounds in userDefinedVarConstraints
- const RealVector & inactive_continuous_lower_bounds () const
  return the inactive continuous lower bounds in userDefinedVarConstraints

- void inactive_continuous_lower_bounds (const RealVector &i_c_l_bnds)
  set the inactive continuous lower bounds in userDefinedVarConstraints
- const RealVector & inactive_continuous_upper_bounds () const
  return the inactive continuous upper bounds in userDefinedVarConstraints
- void inactive_continuous_upper_bounds (const RealVector &i_c_u_bnds)
  set the inactive continuous upper bounds in userDefinedVarConstraints
- const IntVector & inactive_discrete_lower_bounds () const
  return the inactive discrete lower bounds in userDefinedVarConstraints
- void inactive_discrete_lower_bounds (const IntVector &i_d_l_bnds)
  set the inactive discrete lower bounds in userDefinedVarConstraints
- const IntVector & inactive_discrete_upper_bounds () const
  return the inactive discrete upper bounds in userDefinedVarConstraints
- void inactive_discrete_upper_bounds (const IntVector &i_d_u_bnds)
  set the inactive discrete upper bounds in userDefinedVarConstraints
- 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
- const RealVector & linear_ineq_constraint_lower_bounds () const
  return the linear inequality constraint lower bounds
- const RealVector & linear_ineq_constraint_upper_bounds () const
  return the linear inequality constraint upper bounds
- const RealMatrix & linear_eq_constraint_coeffs () const
  return the linear equality constraint coefficients
- const RealVector & linear_eq_constraint_targets () const
  return the linear 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)

- 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 IntArray &asv)

portion of compute_response() specific to derived model classes

- virtual void derived_asynch_compute_response (const IntArray &asv)

  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 ResponseList & 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

Protected Attributes

- Variables currentVariables

  the set of current variables used by the model for performing function evaluations

- size_t numGradVars

  the number of active continuous variables (used in the finite difference routines)

- Response currentResponse

  the set of current responses that holds the results of model function evaluations

- size_t numFns

  the number of functions in currentResponse

- VarConstraints userDefinedVarConstraints

  Explicit constraints on variables are maintained in the VarConstraints 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/asv, 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), INTERFACE, or SUBMODEL

Private Member Functions

- Model * get_model (ProblemDescDB &problem_db)

  Used by the envelope to instantiate the correct letter class.

- size_t estimate_derivatives (const IntArray &map_asv, const IntArray &fd_grad_asv, const IntArray &fd_hess_asv, const IntArray &quasi_hess_asv, const IntArray &original_asv, 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 IntArray &original_asv)

  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 IntArray &original_asv, 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 IntArray &original_asv)

  perform quasi-Newton Hessian updates

- void 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, bool &use_est_deriv)

  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 layered

- bool asynchFDFlag

  flags use of estimate_derivatives w/i asynch_compute_response

- bool asynchEvalFlag

  flags asynch evaluations (local or distributed)

- int evaluationCapacity

  capacity for concurrent evaluations supported by the Model

- 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

- VariablesList varsList

  history of vars populated in asynch_compute_response() and used in synchronize().

- List< IntArray > asvList

  if asynchFDFlag is set, transfers asv sets 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

- SizetList 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.

- ResponseList responseList

  used to return a list of responses for asynchronous evaluations. This list has the responses in final concatenated form. The similar list in Interface contains the raw responses.

- 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 normalDistLowerBnds
  normal uncertain variable distribution lower bounds
- RealVector normalDistUpperBnds
  normal uncertain variable distribution upper bounds
- RealVector lognormalMeans
  lognormal uncertain variable means
- RealVector lognormalStdDevs
  lognormal uncertain variable standard deviations
- RealVector lognormalErrFacts
  lognormal uncertain variable error factors
- RealVector lognormalDistLowerBnds
  lognormal uncertain variable distribution lower bounds
- RealVector lognormalDistUpperBnds
  lognormal uncertain variable distribution upper bounds
- RealVector uniformDistLowerBnds
  uniform uncertain variable distribution lower bounds
- RealVector uniformDistUpperBnds
  uniform uncertain variable distribution upper bounds
- RealVector loguniformDistLowerBnds
  loguniform uncertain variable distribution lower bounds
- RealVector loguniformDistUpperBnds
  loguniform uncertain variable distribution upper bounds
- RealVector weibullAlphas
  weibull uncertain variable alphas
- RealVector weibullBetas
  weibull uncertain variable betas
- RealVector weibullDistLowerBnds
  weibull uncertain variable distribution lower bounds
- RealVector weibullDistUpperBnds
  weibull uncertain variable distribution upper bounds
- RealVectorArray histogramBinPairs

histogram uncertain (x,y) bin pairs (continuous linear histogram)

- RealVectorArray histogramPointPairs

  histogram uncertain (x,y) point pairs (discrete histogram)

8.59.1 Detailed Description

Base class for the model class hierarchy.

The Model class is the base class for one of the primary class hierarchies in DAKOTA. The model hierarchy contains a set of variables, an interface, and a set of responses, and an iterator operates on the model to map the variables into responses using the interface. For memory efficiency and enhanced polymorphism, the model hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Model) serves as the envelope and one of the derived classes (selected in Model::get_model()) serves as the letter.

8.59.2 Constructor & Destructor Documentation

8.59.2.1 Model ()

default constructor

The default constructor is used in vector<Model> instantiations and for initialization of Model objects contained in Iterator and derived Strategy classes. modelRep is NULL in this case (a populated problem_db is needed to build a meaningful Model object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

8.59.2.2 Model (ProblemDescDB & *problem_db*)

standard constructor for envelope

Used in model instantiations within strategy constructors. Envelope constructor only needs to extract enough data to properly execute get_model, since Model(BaseConstructor, problem_db) builds the actual base class data for the derived models.

8.59.2.3 Model (const Model & *model*)

copy constructor

Copy constructor manages sharing of modelRep and incrementing of referenceCount.

8.59.2.4 ~Model () [virtual]

destructor

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

8.59.2.5 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).

8.59.3 Member Function Documentation

8.59.3.1 Model operator= (const Model & *model*)

assignment operator

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

8.59.3.2 String local_eval_synchronization () [virtual]

return derived model synchronization setting

SingleModels and HierLayeredModels 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.
- SurrLayeredModels: while asynch evals on approximations will work due to some added bookkeeping, avoiding them is preferable.

Reimplemented in HierLayeredModel, and SingleModel.

8.59.3.3 int local_eval_concurrency () [virtual]

return derived model asynchronous evaluation concurrency

SingleModels and HierLayeredModels redefine this virtual function.

Reimplemented in HierLayeredModel, and SingleModel.

8.59.3.4 void init_communicators (const int & *max_iterator_concurrency*)

allocate communicator partitions for a model

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.

8.59.3.5 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.

8.59.3.6 void estimate_message_lengths ()

estimate messageLengths for a model

This functionality has been pulled out of init_communicators() and defined separately so that it may be used in those cases when messageLengths is needed but model.init_communicators() is not called, e.g., for the master processor in the self-scheduling of a concurrent iterator strategy.

8.59.3.7 Model * get_model (ProblemDescDB & *problem_db*) [private]

Used by the envelope to instantiate the correct letter class.

Used only by the envelope constructor to initialize modelRep to the appropriate derived type, as given by the modelType attribute.

8.59.3.8 size_t estimate_derivatives (const IntArray & *map_asv*, const IntArray & *fd_grad_asv*, const IntArray & *fd_hess_asv*, const IntArray & *quasi_hess_asv*, const IntArray & *original_asv*, 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.

8.59.3.9 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 IntArray & *original_asv*) [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().

8.59.3.10 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 IntArray & *original_asv*, 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.

8.59.3.11 void 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*, bool & *use_est_deriv*) [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 use_est_deriv is set, 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 use_est_deriv is not set, then only map_asv_out is used.

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

- DakotaModel.H
- DakotaModel.C

8.60 MPIPackBuffer Class Reference

Class for packing MPI message buffers.

Public Member Functions

- MPIPackBuffer (int size_=1024)
  Constructor, which allows the default buffer size to be set.
- ~MPIPackBuffer ()
  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 **fbat'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 **fbat**.
- void pack (const bool &data)
  Pack a **bool**.

Protected Member Functions

- void resize (const int newsize)
  Resizes the internal buffer.

Protected Attributes

- char * Buffer

  The internal buffer for packing.

- int Index

  The index into the current buffer.

- int Size

  The total size that has been allocated for the buffer.

8.60.1 Detailed Description

Class for packing MPI message buffers.

A class that provides a facility for packing message buffers using the MPI_Pack facility. The `MPIPackBuffer` class dynamically resizes the internal buffer to contain enough memory to pack the entire object. When deleted, the `MPIPackBuffer` object deletes this internal buffer. This class is based on the Dakota_Version_3_0 version of utilib::PackBuffer from utilib/src/io/PackBuf.[cpp,h]

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

- MPIPackBuffer.H
- MPIPackBuffer.C

8.61 MPIUnpackBuffer Class Reference

Class for unpacking MPI message buffers.

Public Member Functions

- void setup (char *buf_, int size_, bool flag_=false)
  Method that does the setup for the constructors.
- MPIUnpackBuffer ()
  Default constructor.
- MPIUnpackBuffer (int size_)
  Constructor that specifies the size of the buffer.
- MPIUnpackBuffer (char *buf_, int size_, bool flag_=false)
  Constructor that sets the internal buffer to the given array.
- ~MPIUnpackBuffer ()
  Destructor.
- void resize (const int newsize)
  Resizes the internal buffer.
- const char * buf ()
  Returns a pointer to the internal buffer.
- int size ()
  Returns the length of the buffer.
- int curr ()
  Returns the number of bytes that have been unpacked from the buffer.
- void reset ()
  Resets the index of the internal buffer.
- void unpack (int *data, const int num=1)
  Unpack one or more **int's**.
- void unpack (u_int *data, const int num=1)
  Unpack one or more **unsigned int's**.
- void unpack (long *data, const int num=1)
  Unpack one or more **long's**.
- void unpack (u_long *data, const int num=1)

Unpack one or more **unsigned long's**.

- void unpack (short *data, const int num=1)
  Unpack one or more **short's**.
- void unpack (u_short *data, const int num=1)
  Unpack one or more **unsigned short's**.
- void unpack (char *data, const int num=1)
  Unpack one or more **char's**.
- void unpack (u_char *data, const int num=1)
  Unpack one or more **unsigned char's**.
- void unpack (double *data, const int num=1)
  Unpack one or more **double's**.
- void unpack (float *data, const int num=1)
  Unpack one or more **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.*

8.61.1 Detailed Description

Class for unpacking MPI message buffers.

A class that provides a facility for unpacking message buffers using the MPI_Unpack facility. This class is based on the Dakota_Version_3_0 version of utilib::UnPackBuffer from utilib/src/io/PackBuf.[cpp,h]

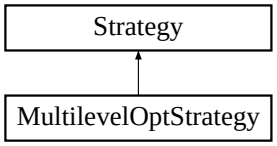
The documentation for this class was generated from the following files:

- MPIPackBuffer.H
- MPIPackBuffer.C

8.62 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 & strategy_variable_results () const
  return the final solution from selectedIterators (variables)
- const Response & strategy_response_results () const
  return the final solution from selectedIterators (response)
- IteratorList & iterators (bool recurse_flag=true)
  returns selectedIterators and any subordinate iterators
- ModelList & models (bool recurse_flag=true)
  returns userDefinedModels and any subordinate models

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

8.62.1 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.

8.62.2 Member Function Documentation

8.62.2.1 void run_coupled () [private]

run a tightly coupled hybrid

In the coupled case, use is made of external hybridization capabilities, such as those available in the global/local hybrids from SGOPT. This function is responsible only for publishing the local optimizer selection to the global optimizer and then invoking the global optimizer; the logic of method switching is handled entirely within the global optimizer. Status: incomplete.

8.62.2.2 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 (using iterator.run_iterator()). Status: fully operational.

8.62.2.3 void run_uncoupled_adaptive () [private]

run an uncoupled adaptive hybrid

In the uncoupled adaptive case, there is interference with the iterators through the use of the ++ overloaded operator. iterator++ runs the iterator for one cycle, after which a progress_metric is computed. This progress metric is used to dictate method switching instead of each iterator's internal convergence criteria. Status: incomplete.

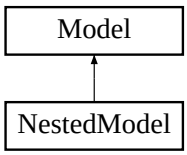
The documentation for this class was generated from the following files:

- MultilevelOptStrategy.H
- MultilevelOptStrategy.C

8.63 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 IntArray &asv)
  portion of compute_response() specific to NestedModel
- void derived_asynch_compute_response (const IntArray &asv)
  portion of asynch_compute_response() specific to NestedModel
- const ResponseArray & derived_synchronize ()
  portion of synchronize() specific to NestedModel
- const ResponseList & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to NestedModel
- const IntList & synchronize_nowait_completions ()
  Return completion id's matching response list from synchronize_nowait.
- Model subordinate_model ()
  return subModel
- Iterator subordinate_iterator ()
  return subIterator
- Interface & interface ()

return optionalInterface

- void layering_bypass (bool bypass_flag)

  NestedModels have nothing to bypass, but must pass request on to the subModel for any lower-level layerings.

- 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 reset_communicators ()

  reset communicator partitions for the NestedModel (forwarded to optionalInterface and subModel)

- void free_communicators ()

  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 total_eval_counter () const

  Return the total evaluation count for the NestedModel.

- int new_eval_counter () const

  Return the new evaluation count for the NestedModel.

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 objLSqCoeffs/constrCoeffs mappings to create the total response for the model

- void update_sub_model ()

  update subModel with current variable values/bounds/labels

Private Attributes

- int nestedEvals

  number of calls to derived_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 interfacePointer

  the optional interface pointer from the nested model specification

- Response interfaceResponse

  the response object resulting from optional interface evaluations

- size_t numInterfObjLSq

  number of objective functions/least squares terms resulting from optional interface evaluations

- size_t numInterfIneqCon

  number of inequality constraints resulting from optional interface evaluations

- size_t numInterfEqCon

  number of equality constraints resulting from the optional interface evaluations

- IntArray 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.

- IntArray 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.

- IntArray 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 objLSqCoeffs

  "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 terms.

- RealMatrix constrCoeffs

  "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.

- ResponseArray responseArray

  dummy response array for derived_synchronize() prior to derived_asynch_compute_response() support

- ResponseList responseList

  dummy response list for derived_synchronize_nowait() prior to derived_asynch_compute_response() support

- IntList completionList

  dummy completion list for synchronize_nowait_completions() prior to derived_asynch_compute_response() support

8.63.1 Detailed Description

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

The NestedModel class nests a sub-iterator execution within every model evaluation. This capability is most commonly used for optimization under uncertainty, in which a nondeterministic iterator is executed on every optimization function evaluation. The NestedModel also contains an optional interface, for portions of the model evaluation which are independent from the sub-iterator, and a set of mappings for combining sub-iterator and optional interface data into a top level response for the model.

8.63.2 Member Function Documentation

8.63.2.1 void derived_compute_response (const IntArray & *asv*) [protected, virtual]

portion of compute_response() specific to NestedModel

Update subModel's inactive variables with active variables from currentVariables, compute the optional interface and sub-iterator responses, and map these to the total model response.

Reimplemented from Model.

8.63.2.2 void derived_asynch_compute_response (const IntArray & *asv*) [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(), derived_synchronize_nowait(), and synchronize_nowait_completions() are inactive as well).

Reimplemented from Model.

8.63.2.3 const ResponseArray & derived_synchronize () [protected, virtual]

portion of synchronize() specific to NestedModel

Asynchronous response computations are not currently supported by NestedModels. Return a dummy responseArray to satisfy the compiler.

Reimplemented from Model.

8.63.2.4 const ResponseList & derived_synchronize_nowait () [protected, virtual]

portion of synchronize_nowait() specific to NestedModel

Asynchronous response computations are not currently supported by NestedModels. Return a dummy responseList to satisfy the compiler.

Reimplemented from Model.

8.63.2.5 const IntList & synchronize_nowait_completions () [inline, protected, virtual]

Return completion id's matching response list from synchronize_nowait.

Asynchronous response computations are not currently supported by NestedModels. Return a dummy completionList to satisfy the compiler.

Reimplemented from Model.

8.63.2.6 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_iterator().

Reimplemented from Model.

8.63.2.7 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.

8.63.2.8 void response_mapping (const Response & *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 objLSqCoeffs/constrCoeffs mappings to create the total response for the model

In the OUU case,

```
optionalInterface fns = {f}, {g} (deterministic obj fns/lsq terms & 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 (objLSqCoeffs class attribute), [A] is the secondary_-mapping_matrix user input (constrCoeffs 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 fns/lsq terms) overlap but optionalInterface/sub-Iterator secondary fns (ineq/eq constraints) do not. The [W] matrix can be specified so as to allow

- some purely deterministic primary functions and some combined: [W] filled and [W].num_rows() < {f}.length() [combined first] *or* [W].num_rows() == {f}.length() and [W] contains rows of zeros [combined last]
- some combined and some purely stochastic primary functions: [W] filled and [W].num_rows() > {f}.length()
- separate deterministic and stochastic primary functions: [W].num_rows() > {f}.length() and [W] contains {f}.length() rows of zeros.

If the need arises, could change constraint definition to allow overlap as well: {g_l} <= {g} + [A]{S} <= {g_u} with [A] usage the same as for [W] above.

In the UOO case, things are simpler, just compute statistics of each optimization response function: [W] = [I], {f}/{g}/[A] are empty.

8.63.3 Member Data Documentation

8.63.3.1 Model subModel [private]

the sub-model used in sub-iterator evaluations

There are no restrictions on subModel, so arbitrary nestings are possible. This is commonly used to support surrogate-based optimization under uncertainty by having NestedModels contain LayeredModels and vice versa.

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

- NestedModel.H
- NestedModel.C

8.64 Nl2Misc Struct Reference

Auxiliary information passed to calcr and calcj via ur.

Public Attributes

- Model * m
  Dakota "Model".
- Real * J [2]
  cache the two most recent Jacobian values in speculative-evaluation mode
- int nf [2]
  function-evaluation counts corresponding to cached Jacobian values (used to tell which J value to use)
- int specgrad
  whether to cache J values (0 == no, 1 == yes)

8.64.1 Detailed Description

Auxiliary information passed to calcr and calcj via ur.

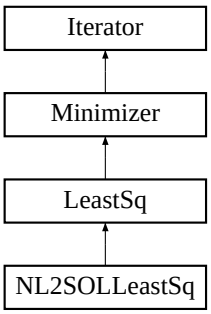
The documentation for this struct was generated from the following file:

- NL2SOLLeastSq.C

8.65 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 ()

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`*)*

8.65.1 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.

8.65.2 Member Function Documentation

8.65.2.1 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.

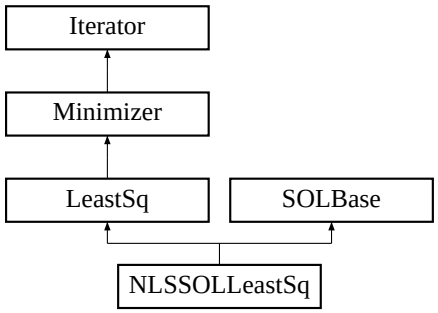
The documentation for this class was generated from the following files:

- NL2SOLLeastSq.H
- NL2SOLLeastSq.C

8.66 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

- 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

- NLSSOLLeastSq * nlssolInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

8.66.1 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 non-static attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows: `max_function_evaluations` is implemented directly in NLSSOLLeastSq's evaluator functions since there is no NLSSOL parameter equivalent, and `max_iterations`, `convergence_tolerance`, `output` verbosity, `verify_level`, `function_precision`, and `linesearch_tolerance` are mapped into NLSSOL's "Major Iteration Limit", "Optimality Tolerance", "Major Print Level" (`verbose:` Major Print Level = 20; `quiet:` Major Print Level = 10), "Verify Level", "Function Precision", and "Linesearch Tolerance" parameters, respectively, using NLSSOL's npoptn() subroutine (as wrapped by npoptn2() from the npoptn_wrapper.f file). Refer to [Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., 1986] for information on NLSSOL's optional input parameters and the npoptn() subroutine.

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

- NLSSOLLeastSq.H
- NLSSOLLeastSq.C

8.67 NoDBBaseConstructor Struct Reference

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

Public Member Functions

- NoDBBaseConstructor (int=0)

  C++ structs can have constructors.

8.67.1 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 (rather than in a header of a class that uses it) avoids problems with circular dependencies.

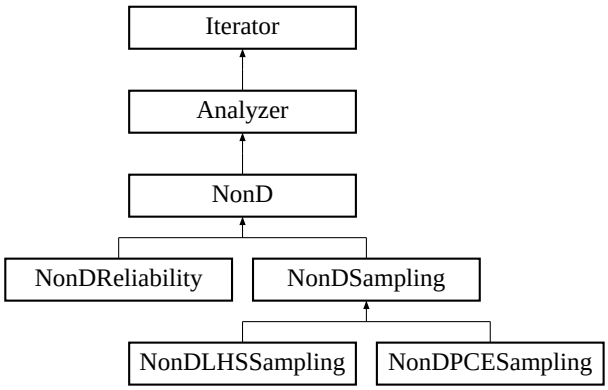
The documentation for this struct was generated from the following file:

- ProblemDescDB.H

8.68 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, int num_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)

  alternate constructor for instantiations "on the fly"

- ~NonD ()

  destructor

- void run_iterator ()

  redefines the main iterator hierarchy virtual function to invoke quantify_uncertainty

- const Response & iterator_response_results () const

  return the final statistics from the nondeterministic iteration

- 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 numWeibullVars

  number of weibull uncertain variables

- size_t numHistogramVars

  number of histogram 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 mean95CIDeltas

  Plus/minus deltas on response function means for 95% confidence intervals (calculated in compute_-statistics()).

- RealVector stdDevStats

  std deviations of response functions (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()).

- 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.

8.68.1 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.

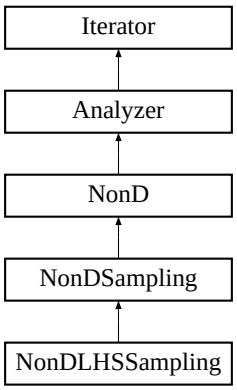
The documentation for this class was generated from the following files:

- DakotaNonD.H
- DakotaNonD.C

8.69 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, int num_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)

  alternate constructor for instantiations "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_iterator_results (ostream &s) const

  print the final statistics

Private Attributes

- bool allVarsFlag

  flags DACE mode using all variables

- bool varBasedDecompFlag

  flags computation of VBD

8.69.1 Detailed Description

Performs LHS and Monte Carlo sampling for uncertainty quantification.

The Latin Hypercube Sampling (LHS) package from Sandia Albuquerque's Risk and Reliability organization provides comprehensive capabilities for Monte Carlo and Latin Hypercube sampling within a broad array of user-specified probabilistic parameter distributions. It enforces user-specified rank correlations through use of a mixing routine. The NonDLHSSampling class provides a C++ wrapper for the LHS library and is used for performing forward propagations of parameter uncertainties into response statistics.

8.69.2 Constructor & Destructor Documentation

8.69.2.1 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.

8.69.2.2 NonDLHSSampling (Model & *model*, int *samples*, int *seed*, int *num_vars*, const RealVector & *lower_bnds*, const RealVector & *upper_bnds*)

alternate constructor for instantiations "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). Data attributes taken from the model in the NoDBBaseConstructor constructors for NonD and Iterator are not used, and other data attributes are not initialized and should not be avoided.

8.69.3 Member Function Documentation

8.69.3.1 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.

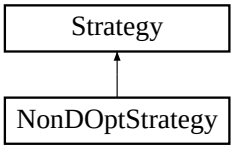
The documentation for this class was generated from the following files:

- NonDLHSSampling.H
- NonDLHSSampling.C

8.70 NonDOptStrategy Class Reference

Strategy for optimization under uncertainty (robust and reliability-based design).

Inheritance diagram for NonDOptStrategy::



Public Member Functions

- NonDOptStrategy (ProblemDescDB &problem_db)

  constructor

- ~NonDOptStrategy ()

  destructor

- void run_strategy ()

  Perform the strategy by executing optIterator (an optimizer) on designModel (a layered or nested model containing a nondeterministic iterator at a lower level).

- const Variables & strategy_variable_results () const

  return the final solution from optIterator (variables)

- const Response & strategy_response_results () const

  return the final solution from optIterator (response)

- IteratorList & iterators (bool recurse_flag=true)

  returns optIterator and any subordinate iterators

- ModelList & models (bool recurse_flag=true)

  returns designModel and any subordinate models

Private Attributes

- Model designModel

  the nested or layered model interfaced with optIterator

- Iterator optIterator

  the top level optimizer

8.70.1 Detailed Description

Strategy for optimization under uncertainty (robust and reliability-based design).

This strategy uses a NestedModel to nest an uncertainty quantification iterator within an optimization iterator in order to perform optimization using nondeterministic data. For OUU based on surrogates, LayeredModels are also employed, and the general recursion facilities supported by nested and layered models allow a broad array of OUU formulations. This class is very simple and is essentially identical to SingleMethodStrategy since all of the nested iteration mappings are contained within NestedModel::response_mapping().

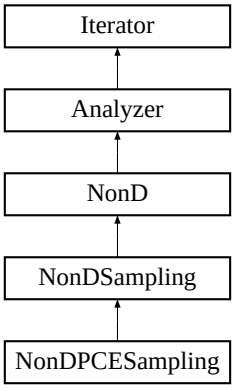
The documentation for this class was generated from the following files:

- NonDOptStrategy.H
- NonDOptStrategy.C

8.71 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_iterator_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.

8.71.1 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.

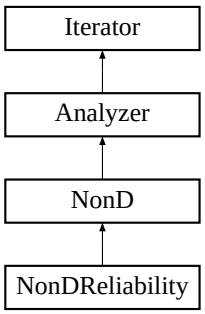
The documentation for this class was generated from the following files:

- NonDPCESampling.H
- NonDPCESampling.C

8.72 NonDReliability Class Reference

Class for the analytical 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_iterator_results (ostream &s) const

  print the approximate mean,standard deviation, and importance factors when using the mean value method (MV) or the CDF information when using other reliability methods (AMV,AMV+,FORM)

- 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 mean_value ()

  convenience function for encapsulating the simple Mean Value computation of approximate statistics and importance factors

- void iterated_mean_value ()

convenience function for encapsulating the iterated reliability methods (AMV, AMV+, FORM, SORM)

- 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 g_eval (int &mode, const Epetra_SerialDenseVector &u, Real &g)

  convenience function for evaluating G(u) and fnGradU(u). Used by RIA_constraint_eval() and both PMA_objective_eval() implementations.

- void transUToX (const Epetra_SerialDenseVector &uncorr_normal_vars, Epetra_SerialDenseVector &random_vars)

  Transformation Routine from u-space of random variables to x-space of random variables for Petra data types.

- void transUToX (const RealVector &uncorr_normal_vars, RealVector &random_vars)

  Transformation Routine from u-space of random variables to x-space of random variables for RealVector data types.

- void transUToZ (const Epetra_SerialDenseVector &uncorr_normal_vars, Epetra_SerialDenseVector &correlated_normal_vars)

  Transformation Routine from u-space of random variables to z-space of random variables for Petra data types.

- void transZToX (const Epetra_SerialDenseVector &correlated_normal_vars, Epetra_SerialDenseVector &random_vars)

  Transformation Routine from z-space of random variables to x-space of random variables for Petra data types.

- void transXToU (const Epetra_SerialDenseVector &random_vars, Epetra_SerialDenseVector &uncorr_normal_vars)

  Transformation Routine from x-space of random variables to u-space of random variables for Petra data types.

- void transXToZ (const Epetra_SerialDenseVector &random_vars, Epetra_SerialDenseVector &correlated_normal_vars)

  Transformation Routine from x-space of random variables to z-space of random variables for Petra data types.

- void transZToU (Epetra_SerialDenseVector &correlated_normal_vars, Epetra_SerialDenseVector &uncorr_normal_vars)

  Transformation Routine from z-space of random variables to u-space of random variables for Petra data types.

- void jacXToU (const Epetra_SerialDenseVector &random_vars, Epetra_SerialDenseMatrix &jacobianXU)

  Jacobian of mapping from x to u random variable space.

- void jacXToZ (const Epetra_SerialDenseVector &random_vars, Epetra_SerialDenseMatrix &jacobianXZ)

  Jacobian of mapping from x to z random variable space.

- void jacUToX (const Epetra_SerialDenseVector &uncorr_normal_vars, Epetra_SerialDenseMatrix &jacobianUX)

  Jacobian of mapping from u to x random variable space.

- void jacZToX (const Epetra_SerialDenseVector &correlated_normal_vars, Epetra_SerialDenseMatrix &jacobianZX)

  Jacobian of mapping from z to x random variable space.

- void transNataf (Epetra_SerialSymDenseMatrix &mod_corr_matrix)

  This procedure modifys the correlation matrix input by the user to be used in the Nataf distribution model.

- double phi (const double &beta)

  Standard normal cumulative distribution function.

- double phi_inverse (const double &p)

  Inverse of standard normal cumulative distribution function.

- double erf_inverse (const double &p)

  Inverse of error function used in phi_inverse().

Static Private Member Functions

- void RIA_objective_eval (int &mode, int &n, Real *u, Real &f, Real *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.

- void RIA_constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int *needc, Real *u, Real *c, Real *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.

- void PMA_objective_eval (int &mode, int &n, Real *u, Real &f, Real *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).

- void PMA_constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int *needc, Real *u, Real *c, Real *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.

- void RIA_objective_eval (int mode, int n, const ColumnVector &u, Real &f, 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.

- void RIA_constraint_eval (int mode, int n, const ColumnVector &u, ColumnVector &g,::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.

- void PMA_objective_eval (int mode, int n, const ColumnVector &u, Real &f, 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).

- void PMA_constraint_eval (int mode, int n, const ColumnVector &u, ColumnVector &g,::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

- size_t numRelAnalyses

  number of invocations of quantify_uncertainty()

- Epetra_SerialDenseVector fnValsMeanX

  copy of response fn values evaluated at mean x

- Epetra_SerialDenseMatrix fnGradsMeanX

  copy of response fn gradients evaluated at mean x

- Epetra_SerialDenseVector fnGradX

  gradient of current response function in x-space

- Epetra_SerialDenseVector fnGradU

  gradient of current response function in u-space

- RealVector medianFnVals

  vector of median values of functions used to determine which side of probability equal 0.5 the response level is

- Epetra_SerialSymDenseMatrix petraCorrMatrix

  petra copy of uncertainCorrelations

- Epetra_SerialDenseMatrix cholCorrMatrix

  cholesky factor of petraCorrMatrix

- 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 mostProbPointULev0

  array of converged MPP's in u-space for level 0. Used for warm-starting of reliability analyses within strategies such as nested RBDO.

- IntVector 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

- int respFnCount

  counter for which response function is being analyzed

- int levelCount

  counter for which response/probability level is being analyzed

- Real requestedRespLevel

  the response 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 computedProbLevel

  output probability level calculated

- Real computedRelLevel

  output reliability level calculated

- short mppSearchFlag

  flag representing the MPP search type selection (MV, AMV, transformed AMV, AMV+, transformed 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

- String integrationMethod

  integration method identifier provided by integration specification

- 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).

- Real Pi

  the value for Pi used in several numerical routines

Static Private Attributes

- NonDReliability * nondRelInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

8.72.1 Detailed Description

Class for the analytical reliability methods within DAKOTA/UQ.

The NonDReliability class implements the following analytic reliability methods: advanced mean value method (AMV), iterated advanced mean value method (AMV+), first order reliability method (FORM), and second order reliability method (SORM). Each of these employ an optimizer (currently NPSOL) to perform a search for the most probable point (MPP).

8.72.2 Member Function Documentation

8.72.2.1 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

Initialize/warm-start optimizer initial guess (initialPtU), linearization point (mostProbPointX/U), and associated response data (computedRespLevel and fnGradX/U).

8.72.2.2 void transUToX (const Epetra_SerialDenseVector & *uncorr_normal_vars*, Epetra_SerialDenseVector & *random_vars*) [private]

Transformation Routine from u-space of random variables to x-space of random variables for Petra data types.

This procedure performs the transformation from u to x space. uncorr_normal_vars is the vector of random variables in standard normal space (u-space). random_vars is the vector of the random variables in the user-defined x-space

8.72.2.3 void transUToZ (const Epetra_SerialDenseVector & *uncorr_normal_vars*, Epetra_SerialDenseVector & *correlated_normal_vars*) [private]

Transformation Routine from u-space of random variables to z-space of random variables for Petra data types.

This procedure computes the transformation from u to z space. uncorr_normal_vars is the vector of random variables in standard normal space (u-space). correlated_normal_vars is the vector of random variables in normal space with proper correlations (z-space).

8.72.2.4 void transZToX (const Epetra_SerialDenseVector & *correlated_normal_vars*, Epetra_SerialDenseVector & *random_vars*) [private]

Transformation Routine from z-space of random variables to x-space of random variables for Petra data types.

This procedure computes the transformation from z to x space. correlated_normal_vars is the vector of random variables in normal space with proper correlations (z-space). random_vars is the vector of the random variables in the user-defined x-space

8.72.2.5 void transXToU (const Epetra_SerialDenseVector & *random_vars*, Epetra_SerialDenseVector & *uncorr_normal_vars*) [private]

Transformation Routine from x-space of random variables to u-space of random variables for Petra data types.

This procedure performs the transformation from x to u space uncorr_normal_vars is the vector of random variables in standard normal space (u-space). random_vars is the vector of the random variables in the user-defined x-space.

8.72.2.6 void transXToZ (const Epetra_SerialDenseVector & *random_vars*, Epetra_SerialDenseVector & *correlated_normal_vars*) [private]

Transformation Routine from x-space of random variables to z-space of random variables for Petra data types.

This procedure performs the transformation from x to z space: correlated_normal_vars is the vector of random variables in normal space with proper correlations(z-space). random_vars is the vector of the random variables in the user-defined x-space.

8.72.2.7 void transZToU (Epetra_SerialDenseVector & *correlated_normal_vars*, Epetra_SerialDenseVector & *uncorr_normal_vars*) [private]

Transformation Routine from z-space of random variables to u-space of random variables for Petra data types.

This procedure computes the transformation from z to u space. uncorr_normal_vars is the vector of random variables in standard normal space (u-space). correlated_normal_vars is the vector of random variables in normal space with proper correlations (z-space).

8.72.2.8 void jacXToU (const Epetra_SerialDenseVector & *random_vars*, Epetra_SerialDenseMatrix & *jacobianXU*) [private]

Jacobian of mapping from x to u random variable space.

This procedure computes the jacobian of the transformation from x to u space. random_vars is the vector of the random variables in the user-defined x-space.

8.72.2.9 void jacXToZ (const Epetra_SerialDenseVector & *random_vars*, Epetra_SerialDenseMatrix & *jacobianXZ*) [private]

Jacobian of mapping from x to z random variable space.

This procedure computes the jacobian of the transformation from x to z space. random_vars is the vector of the random variables in the user-defined x-space.

8.72.2.10 void jacUToX (const Epetra_SerialDenseVector & *uncorr_normal_vars*, Epetra_SerialDenseMatrix & *jacobianUX*) [private]

Jacobian of mapping from u to x random variable space.

This procedure computes the jacobian of the transformation from u to x space. uncorr_normal_vars is the vector of random variables in standard normal space (u-space).

8.72.2.11 void jacZToX (const Epetra_SerialDenseVector & *correlated_normal_vars*, Epetra_SerialDenseMatrix & *jacobianZX*) [private]

Jacobian of mapping from z to x random variable space.

This procedure computes the jacobian of the transformation from z to x space. correlated_normal_vars is the vector of random variables in normal space with proper correlations (z-space).

8.72.2.12 void transNataf (Epetra_SerialSymDenseMatrix & *mod_corr_matrix*) [private]

This procedure modifys the correlation matrix input by the user to be used in the Nataf distribution model.

This procedure modifys the correlation matrix input by the user to be used in the Nataf distribution model (der Kiureghian and Liu, ASCE JEM 112:1, 1986).

R: the correlation coefficient matrix of the random variables

mod_corr_matrix: modified correlation matrix

Note: The modification is exact for log-log,normal-log,normal-normal, normal-uniform tranformations (numerical precision). The uniform-uniform and uniform-log case are approximations obtained in the above reference.

8.72.2.13 double phi (const double & *beta*) [private]

Standard normal cumulative distribution function.

returns a probability < 0.5 for negative beta and a probability > 0.5 for positive beta.

8.72.2.14 double phi_inverse (const double & *p*) [private]

Inverse of standard normal cumulative distribution function.

returns a negative beta for probability < 0.5 and a positive beta for probability > 0.5.

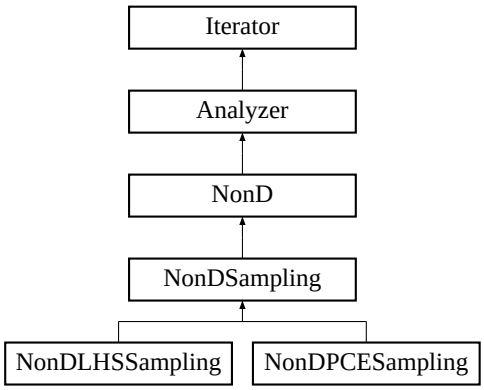
The documentation for this class was generated from the following files:

- NonDReliability.H
- NonDReliability.C

8.73 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, int num_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)

  alternate constructor for instantiations "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 get_parameter_sets (bool vbd_change_seq_flag)

  Uses run_lhs() to generate a set of samples. In the usual mode, this will be called once. In variance-based decomposition or replicated LHS, it may be called several times.

- void run_lhs ()

  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

- bool rank_sort (const int &x, const int &y)

  sort algorithm to compute ranks for rank correlations

Protected Attributes

- int numObservations

  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

- 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

- RealArray rawData

  vector to hold raw data before rank sort

- int pgf90Initialized

  flag indicating whether pghpf_init() has been called.

8.73.1 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.

8.73.2 Constructor & Destructor Documentation

8.73.2.1 NonDSampling (Model & *model*) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

8.73.2.2 NonDSampling (NoDBBaseConstructor, Model & *model*, int *samples*, int *seed*, int *num_vars*, const RealVector & *lower_bnds*, const RealVector & *upper_bnds*) [protected]

alternate constructor for instantiations "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets.

8.73.3 Member Function Documentation

8.73.3.1 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.

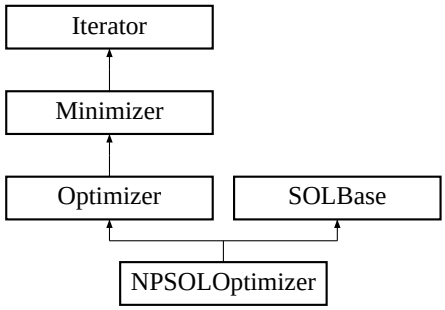
The documentation for this class was generated from the following files:

- NonDSampling.H
- NonDSampling.C

8.74 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, int num_lin_ineq, int num_lin_eq, int num_nln_ineq, int num_nln_-eq, 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 &, Real ∗, Real &, Real ∗, int &), void(∗user_con_eval)(int &, int &, int &, int &, int ∗, Real ∗, Real ∗, Real ∗, 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

- 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 &, Real *, Real &, Real *, int &)

  holds function pointer for objective function evaluator passed in for "user_functions" mode.

- void(* userConstraintEval)(int &, int &, int &, int &, int *, Real *, Real *, Real *, int &)

  holds function pointer for constraint function evaluator passed in for "user_functions" mode.

Static Private Attributes

- NPSOLOptimizer * npsolInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

8.74.1 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.

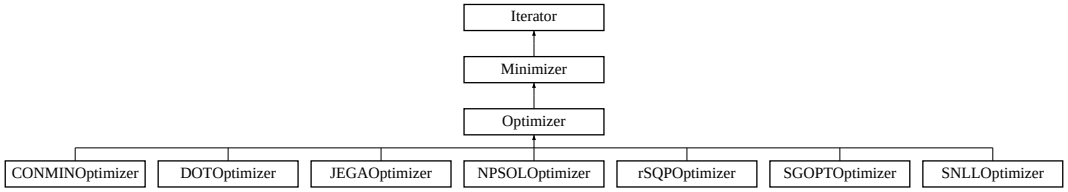
The documentation for this class was generated from the following files:

- NPSOLOptimizer.H
- NPSOLOptimizer.C

8.75 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy.

Inheritance diagram for Optimizer::



Public Member Functions

- void run_iterator ()

  run the iterator

Protected Member Functions

- Optimizer ()

  default constructor

- Optimizer (Model &model)

  standard constructor

- ~Optimizer ()

  destructor

- void print_iterator_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.

- 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

8.75.1 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 SGOPTOptimizer.

8.75.2 Constructor & Destructor Documentation

8.75.2.1 Optimizer (Model & *model*) [protected]

standard constructor

This constructor extracts the inherited data for the optimizer branch and performs sanity checking on gradient and constraint settings.

8.75.3 Member Function Documentation

8.75.3.1 void run_iterator () [inline, virtual]

run the iterator

This function is the primary run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

8.75.3.2 void print_iterator_results (ostream & *s*) const [protected, virtual]

Redefines default iterator results printing to include optimization results (objective function and constraints).

Reimplemented from Iterator.

8.75.3.3 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.

8.75.3.4 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

8.76 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)

8.76.1 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

8.77 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

8.77.1 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

8.78 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

- 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

- void decrement_parallel_configuration ()

  decrement currPCIter

- 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

8.78.1 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.

8.78.2 Constructor & Destructor Documentation

8.78.2.1 ParallelLibrary (int & *argc*, char **& *argv*)

stand-alone mode constructor

This constructor is the one used by main.C. It calls MPI_Init conditionally based on whether a parallel launch is detected.

8.78.2.2 ParallelLibrary ()

library mode constructor

This constructor provides a library mode and is used by the SIERRA Adak application. It does not call MPI_Init, but rather gathers data from MPI_COMM_WORLD if MPI_Init has been called elsewhere.

8.78.2.3 ParallelLibrary (int *dummy*)

dummy constructor (used for dummy_lib)

This constructor is used for creation of the global dummy_lib object, which is used to satisfy initialization requirements when the real ParallelLibrary object is not available.

8.78.3 Member Function Documentation

8.78.3.1 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.

8.78.3.2 void manage_outputs_restart (const ParallelLevel & *pl*)

manage output streams and restart file(s) (both modes)

If the user has specified the use of files for DAKOTA standard output and/or standard error, then bind these filenames to the Cout/Cerr macros. In addition, if concurrent iterators are to be used, create and tag multiple output streams in order to prevent jumbled output. Manage restart file(s) by processing any incoming evaluations from an old restart file and by setting up the binary output stream for new evaluations. Only master iterator processor(s) read & write restart information. This function must follow init_iterator_communicators so that restart can be managed properly for concurrent iterator strategies. In the case of concurrent iterators, each iterator has its own restart file tagged with iterator number.

8.78.3.3 void close_streams ()

close streams, files, and any other services

Close streams associated with manage_outputs and manage_restart and terminate any additional services that may be active.

8.78.3.4 void init_communicators (const ParallelLevel & *parent_pl*, const int & *num_servers*, const int & *procs_per_server*, const int & *max_concurrency*, const int & *asynch_local_concurrency*, const String & *default_config*, const String & *scheduling_override*) [private]

split a parent communicator into child server communicators

Split parent communicator into concurrent child server partitions as specified by the passed parameters. This constructs new child intra-communicators and parent-child inter-communicators. This function is called from the Strategy constructor for the concurrent iterator level and from ApplicationInterface::init_communicators() for the concurrent evaluation and concurrent analysis levels.

8.78.3.5 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

8.79 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 Response &response)
  alternate constructor for temporaries
- ParamResponsePair (const Variables &vars, const Response &response, const int id)
  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 IntArray & active_set_vector () const

  return the active set vector from the response object

- void active_set_vector (const IntArray &asv)

  set the active set vector in 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

- 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

8.79.1 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 this class with the pair<> template construct may be possible (using pair<int, pair<vars,response> >, for example), assuming that deep copies, I/O, alternate constructors, etc., can be adequately addressed.

8.79.2 Constructor & Destructor Documentation

8.79.2.1 ParamResponsePair (const Variables & *vars*, const Response & *response*) [inline]

alternate constructor for temporaries

This constructor can use the standard Variables and Response copy constructors to share representations since this constructor is used for search_pairs (which are local instantiations that go out of scope prior to any changes to values; i.e., they are not used for history).

8.79.2.2 ParamResponsePair (const Variables & *vars*, const Response & *response*, const int *id*) [inline]

standard constructor for history uses

This constructor cannot share representations since it involves a history mechanism (beforeSynchPRPList or data_pairs). Deep copies must be made.

8.79.3 Member Data Documentation

8.79.3.1 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, so evalId is meaningful.

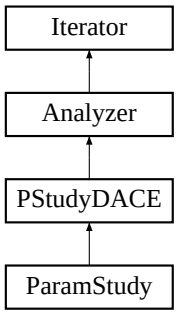
The documentation for this class was generated from the following files:

- ParamResponsePair.H
- ParamResponsePair.C

8.80 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 (pStudy-Type 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)

8.80.1 Detailed Description

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

The ParamStudy class contains several algorithms for performing parameter studies of different types. It is not a wrapper for an external library, rather its algorithms are self-contained. The vector parameter study steps along an n-dimensional vector from an arbitrary initial point to an arbitrary final point in a specified number of steps. The centered parameter study performs a number of plus and minus offsets in each coordinate direction around a center point. A multidimensional parameter study fills an n-dimensional hypercube based on a specified number of intervals for each dimension. It is a nested study in that it utilizes the vector parameter study internally as it recurses through the variables. And the list parameter study provides for a user specification of a list of points to evaluate, which allows general parameter investigations not fitting the structure of vector, centered, or multidim parameter studies.

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

- ParamStudy.H
- ParamStudy.C

8.81 ProblemDescDB Class Reference

The database containing information parsed from the DAKOTA input file.

Public Member Functions

- ProblemDescDB (ParallelLibrary &parallel_lib)

  constructor

- ~ProblemDescDB ()

  destructor

- 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 methodIter based on the method identifier string to activate a particular method specification in methodList and use pointers from this method specification to set the other list iterators.

- void set_db_list_nodes (const size_t &method_index)

  set methodIter based on the active index to activate a particular method specification in methodList and use pointers from this method specification to set the other list iterators.

- size_t get_db_list_nodes ()

  return the index of the active node in methodList

- void set_db_interface_node (const String &interface_tag)

  set interfaceIter based on the interface identifier string

- void set_db_responses_node (const String &responses_tag)

  set responsesIter based on the responses identifier string

- void set_db_model_type (const String &model_type)

  set the model type

- ParallelLibrary & parallel_library () const

  return the parallelLib reference

- 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 methodList
- void insert_node (const DataVariables &data_variables)
  add a DataVariables object to the variablesList

- void insert_node (const DataInterface &data_interface)

  add a DataInterface object to the interfaceList

- void insert_node (const DataResponses &data_responses)

  add a DataResponses object to the responsesList

Static Public Member Functions

- void method_kwhandler (const struct FunctionData *parsed_data)

  method keyword handler called by IDR when a complete method specification is parsed

- void variables_kwhandler (const struct FunctionData *parsed_data)

  variables keyword handler called by IDR when a complete variables specification is parsed

- void interface_kwhandler (const struct FunctionData *parsed_data)

  interface keyword handler called by IDR when a complete interface specification is parsed

- void responses_kwhandler (const struct FunctionData *parsed_data)

  responses keyword handler called by IDR when a complete responses specification is parsed

- void strategy_kwhandler (const struct FunctionData *parsed_data)

  strategy keyword handler called by IDR when a complete strategy specification is parsed

Private Member Functions

- void send_db_buffer ()

  MPI send of a large buffer containing strategy specification attributes and all the objects in interfaceList, variablesList, methodList, and responsesList. Used by manage_inputs().

- void receive_db_buffer ()

  MPI receive of a large buffer containing strategy specification attributes and all the objects in interfaceList, variablesList, methodList, and responsesList. Used by manage_inputs().

- void build_label (String &label, const String &root_label, size_t tag)

  create a label by appending tag to 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().

- void set_other_list_nodes ()

  convenience function used by set_db_list_nodes(method_tag) and set_db_list_nodes(method_index) to set the other list iterators once methodIter is set (based on pointers from the method specification).

Static Private Member Functions

- void idr_kw_id_error (const char *kw)

  Error handler for missing required IDR keyword.

- 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.

- 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.

- 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.

- 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.

Private Attributes

- ParallelLibrary & parallelLib

  reference to the parallel_lib object passed from main

- DataStrategy strategySpec

  the strategy specification (only one allowed) resulting from a call to strategy_kwhandler() or insert_node()

- List< DataMethod > methodList

  list of method specifications, one for each call to method_kwhandler() or insert_node()

- List< DataVariables > variablesList

  list of variables specifications, one for each call to variables_kwhandler() or insert_node()

- List< DataInterface > interfaceList

  list of interface specifications, one for each call to interface_kwhandler() or insert_node()

- List< DataResponses > responsesList

  list of responses specifications, one for each call to responses_kwhandler() or insert_node()

- List< DataMethod >::iterator methodIter

  iterator identifying the active list node in methodList

- List< DataVariables >::iterator variablesIter

  iterator identifying the active list node in variablesList

- List< DataInterface >::iterator interfaceIter

  iterator identifying the active list node in interfaceList

- List< DataResponses >::iterator responsesIter

  iterator identifying the active list node in responsesList

- size_t strategyCntr

  counter for strategy specifications used in check_input

- bool dbLocked

  prevents use of get_<type> data retrieval functions prior to a set_db_list_nodes invocation

- bool dummyFlag

  prevents multiple deallocations for true DB/dummy_db

Static Private Attributes

- ProblemDescDB * pDDBInstance

  pointer to the active object instance used within the static kwhandler functions in order to avoid the need for static data

- Int ** intTable

  integer table populated in idr_get_int_table()

- Real ** realTable

  real table populated in idr_get_real_table()

- char *** stringTable

  string table populated in idr_get_string_table()

8.81.1 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 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 a data class object (DataStrategy, DataMethod, DataVariables, DataInterface, or DataResponses) and, for all cases except strategy, appends the object to a linked list (methodList, variablesList, interfaceList, or responsesList). No strategy linked list is used since only one strategy specification is allowed. For information on modifying the input parsing procedures, refer to Dakota/docs/spec_change_instructions.txt

8.81.2 Member Function Documentation

8.81.2.1 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 using the Input Deck Reader (IDR) parsing system. IDR populates the ProblemDescDB object with the input file data.

8.81.2.2 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 using the Input Deck Reader (IDR) parsing system. IDR populates the ProblemDescDB object with the input file data.

8.81.2.3 void set_db_model_type (const String & *model_type*) [inline]

set the model type

Used to avoid recursion in DakotaModel::get_model() by a sub model when get_string("method.model_-type") is not reset by a sub iterator. Note: if more needs of this type arise, could add set_<type> member functions to parallel the existing get_<type> member functions.

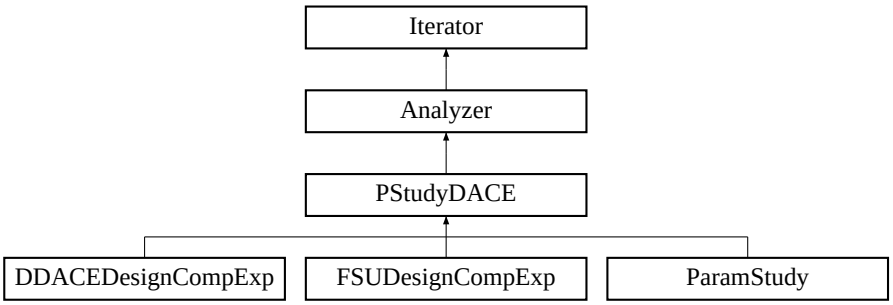
The documentation for this class was generated from the following files:

- ProblemDescDB.H
- ProblemDescDB.C

8.82 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_iterator ()

  run the iterator

- const Variables & iterator_variable_results () const

  return the final iterator solution (variables)

- const Response & iterator_response_results () const

  return the final iterator solution (response)

- void print_iterator_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

- size_t numNonlinIneqCons

  number of nonlinear inequality constraints

- size_t numNonlinEqCons

  number of nonlinear equality constraints

- RealVector multiObjWts

  vector of multiobjective weights

- RealVector nonlinIneqLowerBnds

  vector of nonlinear inequality constraint lower bounds

- RealVector nonlinIneqUpperBnds

  vector of nonlinear inequality constraint upper bounds

- RealVector nonlinEqTargets

  vector of nonlinear equality constraint targets

8.82.1 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.

8.82.2 Member Function Documentation

8.82.2.1 void run_iterator () [inline, protected, virtual]

run the iterator

This function is the primary run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

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

- DakotaPStudyDACE.H
- DakotaPStudyDACE.C

8.83 Response Class Reference

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

Public Member Functions

- Response ()
  default constructor
- Response (int num_params, const ProblemDescDB &problem_db)
  standard constructor built from problem description database
- Response (int num_params, const IntArray &asv)
  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 IntArray & active_set_vector () const
  return the active set vector
- void active_set_vector (const IntArray &asv)
  set the active set vector
- const String & interface_id () const
  return the interface identifier
- void interface_id (const String &id)
  set the interface 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)
  handle class forward to corresponding body class member function
- void purge_inactive ()
  handle class forward to corresponding body class member function
- void reset ()
  handle class forward to corresponding body class member function
- bool is_null () const
  function to check responseRep (does this handle contain a body)

Private Attributes

- ResponseRep * responseRep
  pointer to the body (handle-body idiom)

Friends

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

8.83.1 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.

8.83.2 Constructor & Destructor Documentation

8.83.2.1 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

8.84 ResponseRep Class Reference

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

Private Member Functions

- ResponseRep ()
  default constructor
- ResponseRep (int num_params, const ProblemDescDB &problem_db)
  standard constructor built from problem description database
- ResponseRep (int num_params, const IntArray &asv)
  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 Response &response)

  copy functionValues, functionGradients, & functionHessians data only. Do not copy ASV, tags, id's, etc. Used in place of assignment operator for retrieving results data from the data_pairs list without corrupting other data.

- void purge_inactive ()

  Purge extraneous inactive data from the response object.

- void reset ()

  resets functionValues, functionGradients, and functionHessians to zero

Private Attributes

- int referenceCount

  number of handle objects sharing responseRep

- RealVector functionValues

  abstract set of functions

- RealMatrix functionGradients

  first derivatives

- RealMatrixArray functionHessians

  second derivatives

- IntArray responseASV

  Copy of Iterator::activeSetVector needed for operator overloaded I/O.

- StringArray fnTags

  function identifiers used to improve output readability

- String interfaceId

  the interface used to generate this response object. Used in ParamResponsePair::vars_asv_compare.

Friends

- bool operator== (const ResponseRep &rep1, const ResponseRep &rep2)

  equality operator

8.84.1 Detailed Description

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

The ResponseRep class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (Response) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (ResponseRep) actually contains the response data (functionValues, functionGradients, functionHessians, etc.). The representation is hidden in that an instance of ResponseRep may only be created by Response. Therefore, programmers create instances of the Response handle class, and only need to be aware of the handle/body mechanisms when it comes to managing shallow copies (shared representation) versus deep copies (separate representation used for history mechanisms).

8.84.2 Constructor & Destructor Documentation

8.84.2.1 ResponseRep (int *num_params*, const ProblemDescDB & *problem_db*) [private]

standard constructor built from problem description database

The standard constructor used by Dakota::ModelRep. An interfaceId identifies a set of results with the interface used in generating them, which allows vars_asv_compare to prevent duplicate detection on results from different interfaces.

8.84.2.2 ResponseRep (int *num_params*, const IntArray & *asv*) [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 and interfaceId are not needed for this purpose since they're not passed in the MPI send/recv buffers (NOTE: if interfaceId becomes needed, it could be set from an AppInt attribute passed from AppInt::serve()). 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.

8.84.3 Member Function Documentation

8.84.3.1 void read (istream & *s*) [private]

read a responseRep object from an istream

ASCII version of read needs capabilities for capturing data omissions or formatting errors (resulting from user error or asynch race condition) and analysis failures (resulting from nonconvergence, instability, etc.).

8.84.3.2 void write (ostream & *s*) const [private]

write a responseRep object to an ostream

ASCII version of write.

8.84.3.3 void read_annotated (istream & *s*) [private]

read a responseRep object from an istream (annotated format)

read_annotated version is used for neutral file translation of restart files. Since objects are built solely from this data, annotations are used. This version closely mirrors the BiStream version.

8.84.3.4 void write_annotated (ostream & *s*) const [private]

write a responseRep object to an ostream (annotated format)

write_annotated version is used for neutral file translation of restart files. Since objects need to be build solely from this data, annotations are used. This version closely mirrors the BoStream version, with the exception of the use of white space between fields.

8.84.3.5 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 Approximation-Interfaces in reading samples from a file. There is insufficient data in a tabular file to build complete response objects; rather, the response object must be constructed a priori and then its functionValues can be set.

8.84.3.6 void write_tabular (ostream & *s*) const [private]

write functionValues to an ostream (tabular format)

write_tabular is used for output of functionValues in a tabular format for convenience in post-processing/plotting of DAKOTA results.

8.84.3.7 void read (BiStream & *s*) [private]

read a responseRep object from a binary stream

Binary version differs from ASCII version in 2 primary ways: (1) it lacks formatting. (2) the Response has not been sized a priori. In reading data from the binary restart file, a ParamResponsePair was constructed with its default constructor which called the Response default constructor. Therefore, we must first read sizing data and resize all of the arrays.

8.84.3.8 void write (BoStream & *s*) const [private]

write a responseRep object to a binary stream

Binary version differs from ASCII version in 2 primary ways: (1) It lacks formatting. (2) In reading data from the binary restart file, ParamResponsePairs are constructed with their default constructor which calls the Response default constructor. Therefore, we must first write sizing data so that ResponseRep::read(BoStream& s) can resize the arrays.

8.84.3.9 void read (MPIUnpackBuffer & *s*) [private]

read a responseRep object from a packed MPI buffer

UnpackBuffer version differs from BiStream version in the omission of interfaceId and fnTags. Master processor retains function tags and interface ids and communicates asv and response data only with slaves.

8.84.3.10 void write (MPIPackBuffer & *s*) const [private]

write a responseRep object to a packed MPI buffer

MPIPackBuffer version differs from BoStream version only in omissions of interfaceId and fnTags. The master processor retains tags and ids and communicates asv and response data only with slaves.

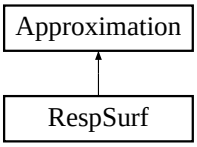
The documentation for this class was generated from the following files:

- DakotaResponse.H
- DakotaResponse.C

8.85 RespSurf Class Reference

Derived approximation class for polynomial regression.

Inheritance diagram for RespSurf::



Public Member Functions

- RespSurf (const ProblemDescDB &problem_db, const size_t &num_acv)
  constructor
- ~RespSurf ()
  destructor

Protected Member Functions

- void find_coefficients ()
  Least squares fit to data using a singular value decomposition.
- int required_samples ()
  return the minimum number of samples required to build the derived class approximation type in numVars dimensions
- const RealVector & approximation_coefficients ()
  return the coefficient array computed by find_coefficients()
- 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

Private Attributes

- int numCoeffs
  number of coefficients used by the polynomial model
- RealVector polyCoeffs

vector of polynomial coefficients

- short polyOrder

  flag to indicate a linear (value = 1), quadratic (value = 2), or cubic (value = 3) polynomial model

8.85.1 Detailed Description

Derived approximation class for polynomial regression.

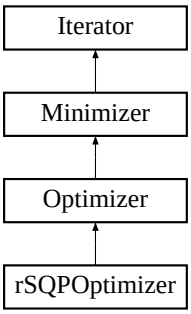
The RespSurf class computes a linear, quadratic, or cubic polynomial fit to data. The polynomial has either n+1 (linear case), (n+1)∗(n+2)/2 (quadratic case), or (n^3+6n^2+11n+6)/6 (cubic case) coefficients for n variables. A least squares estimation of the polynomial coefficients is performed using LAPACK'S linear least squares subroutine DGELSS which uses a singular value decomposition method.

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

- RespSurf.H
- RespSurf.C

8.86 rSQPOptimizer Class Reference

Inheritance diagram for rSQPOptimizer::



Public Member Functions

- **rSQPOptimizer** (Model &model)
- int **num_objectives** () const
- const RealVector & **lin_ineq_lb** () const
- const RealVector & **lin_ineq_ub** () const
- const RealVector & **nonlin_ineq_lb** () const
- const RealVector & **nonlin_ineq_ub** () const
- const RealVector & **lin_eq_targ** () const
- const RealVector & **nonlin_eq_targ** () const
- const RealMatrix & **lin_eq_jac** () const
- const RealMatrix & **lin_ineq_jac** () const

Overridden from Optimizer

- void find_optimum ()

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

Private Attributes

- Model * **model_**
- NLPInterfacePack::NLPDakota **nlp_**

8.86.1 Detailed Description

Wrapper class for the rSQP++ optimization library.

The rSQPOptimizer class provides a wrapper for rSQP++, a C++ sequential quadratic programming library written by Roscoe Bartlett. rSQP++ can currently be used in NAND mode, although use of its SAND

mode for reduced-space SQP is planned. rSQPOptimizer uses a NLPDakota object to perform the function evaluations.

The user input mappings will ultimately include: `max_iterations`, `convergence_tolerance`, `output_verbosity`.

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

- rSQPOptimizer.H
- rSQPOptimizer.C

8.87 SGOPTApplication Class Reference

Maps the evaluation functions used by SGOPT algorithms to the DAKOTA evaluation functions.

Public Member Functions

- SGOPTApplication (SGOPTOptimizer *instance, int type)
  constructor
- ~SGOPTApplication ()
  destructor
- int DoEval (OptPoint &pt, OptResponse *response, int synch_flag)
  launch a function evaluation either synchronously or asynchronously
- int synchronize ()
  blocking retrieval of all pending jobs
- int next_eval (int &id)
  nonblocking query and retrieval of a job if completed
- void dakota_asynch_flag (const bool &asynch_flag)
  set dakotaModelAsynchFlag

Private Member Functions

- void copy (const Response &, OptResponse &)
  copy data from a Response object to an SGOPT OptResponse object

Private Attributes

- SGOPTOptimizer * sgoptOptInstance
  pointer to the SGOPTOptimizer instance for access to optimizer data
- IntArray activeSetVector
  copy/conversion of the SGOPT request vector
- bool dakotaModelAsynchFlag
  a flag for asynchronous DAKOTA evaluations
- ResponseList dakotaResponseList
  list of DAKOTA responses returned by synchronize_nowait()

- IntList dakotaCompletionList

  list of DAKOTA completions returned by synchronize_nowait_completions()

8.87.1 Detailed Description

Maps the evaluation functions used by SGOPT algorithms to the DAKOTA evaluation functions.

SGOPTApplication is a DAKOTA class that is derived from SGOPT's AppInterface hierarchy. It redefines a variety of virtual SGOPT functions to use the corresponding DAKOTA functions. This is a more flexible algorithm library interfacing approach than can be obtained with the function pointer approaches used by NPSOLOptimizer and SNLLOptimizer.

8.87.2 Member Function Documentation

8.87.2.1 int DoEval (OptPoint & *pt*, OptResponse * *prob_response*, int *synch_flag*)

launch a function evaluation either synchronously or asynchronously

Converts SGOPT 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 SGOPT response (synchronous) or bookkeeps the SGOPT response object (asynchronous).

8.87.2.2 int synchronize ()

blocking retrieval of all pending jobs

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

8.87.2.3 int next_eval (int & *id*)

nonblocking query and retrieval of a job if completed

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

8.87.2.4 void dakota_asynch_flag (const bool & *asynch_flag*) [inline]

set dakotaModelAsynchFlag

This function is needed to publish the iterator's asynchFlag at run time (asynchFlag not available at construction).

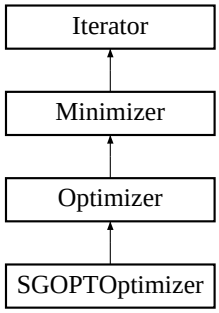
The documentation for this class was generated from the following files:

- SGOPTApplication.H
- SGOPTApplication.C

8.88 SGOPTOptimizer Class Reference

Wrapper class for the SGOPT optimization library.

Inheritance diagram for SGOPTOptimizer::



Public Member Functions

- SGOPTOptimizer (Model &model)
  constructor
- ~SGOPTOptimizer ()
  destructor
- void find_optimum ()
  Performs the iterations to determine the optimal solution.

Private Member Functions

- void set_method_options ()
  sets options for the methods based on user specifications

Private Attributes

- String exploratoryMoves
  user input for desired pattern search algorithm variant
- bool discreteAppFlag
  convenience flag for integer vs. real applications
- PM_LCG * linConGenerator
  Pointer to random number generator.

- BaseOptimizer * baseOptimizer
  Pointer to SGOPT base optimizer object.
- AppInterface * sgoptApplication
  pointer to the SGOPTApplication object
- RealOptProblem * realProblem
  pointer to RealOptProblem object
- IntOptProblem * intProblem
  pointer to IntOptProblem object
- PGAreal * pGARealOptimizer
  pointer to PGAreal object
- PGAint * pGAIntOptimizer
  pointer to PGAint object
- EPSA * ePSAOptimizer
  pointer to EPSA object
- PatternSearch * patternSearchOptimizer
  pointer to PatternSearch object
- APPSOpt * aPPSOptimizer
  pointer to APPSOpt object
- SWOpt * sWOptimizer
  pointer to SWOpt object
- sMCreal * sMCrealOptimizer
  pointer to sMCreal object

8.88.1 Detailed Description

Wrapper class for the SGOPT optimization library.

The SGOPTOptimizer class provides a wrapper for SGOPT, a Sandia-developed C++ optimization library of genetic algorithms, pattern search methods, and other nongradient-based techniques. It uses an SGOPTApplication 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 SGOPT's `max_iters`, `max_neval`, `ftol`, `accuracy`, and `max_time` data attributes. An `output` setting of `verbose` is passed to SGOPT's set_output() function and a setting of `debug` activates output of method initialization and sets the SGOPT `debug` attribute to 10000. SGOPT 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 SGOPTApplication 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 SGOPT objects and controls.

8.88.2 Constructor & Destructor Documentation

8.88.2.1 SGOPTOptimizer (Model & *model*)

constructor

The constructor allocates the objects and populates the class member pointer attributes.

8.88.2.2 ~SGOPTOptimizer ()

destructor

The destructor deallocates the class member pointer attributes.

8.88.3 Member Function Documentation

8.88.3.1 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 SGOPT. It first sets up the problem data, then executes minimize() on the SGOPT algorithm, and finally catalogues the results.

Implements Optimizer.

8.88.3.2 void set_method_options () [private]

sets options for the methods based on user specifications

set_method_options propagates DAKOTA user input to the appropriate SGOPT objects.

8.88.4 Member Data Documentation

8.88.4.1 AppInterface* sgoptApplication [private]

pointer to the SGOPTApplication object

SGOPTApplication is a DAKOTA class derived from the SGOPT AppInterface class. It redefines the virtual SGOPT evaluation functions to use DAKOTA evaluation functions.

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

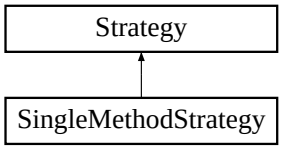
- SGOPTOptimizer.H

- SGOPTOptimizer.C

8.89 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 & strategy_variable_results () const
  return the final solution from selectedIterator (variables)
- const Response & strategy_response_results () const
  return the final solution from selectedIterator (response)
- IteratorList & iterators (bool recurse_flag=true)
  returns selectedIterator and any subordinate iterators
- ModelList & models (bool recurse_flag=true)
  returns userDefinedModel and any subordinate models

Private Attributes

- Model userDefinedModel
  the model to be iterated
- Iterator selectedIterator
  the iterator

8.89.1 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.

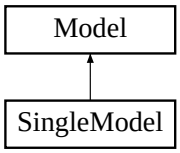
The documentation for this class was generated from the following files:

- SingleMethodStrategy.H
- SingleMethodStrategy.C

8.90 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 IntArray &asv)

  portion of compute_response() specific to SingleModel (invokes a synchronous map() on userDefinedInterface)

- void derived_asynch_compute_response (const IntArray &asv)

  portion of asynch_compute_response() specific to SingleModel (invokes an asynchronous map() on userDefinedInterface)

- const ResponseArray & derived_synchronize ()

  portion of synchronize() specific to SingleModel (invokes synch() on userDefinedInterface)

- const ResponseList & derived_synchronize_nowait ()

  portion of synchronize_nowait() specific to SingleModel (invokes synch_nowait() on userDefinedInterface)

- const IntList & synchronize_nowait_completions ()

  return completion id's matching response list from synchronize_nowait (request forwarded to 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 userDefinedInterface)

- 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 free_communicators ()

  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 total_eval_counter () const

  return the total evaluation count for the SingleModel (request forwarded to userDefinedInterface)

- int new_eval_counter () const

  return the new evaluation count for the SingleModel (request forwarded to userDefinedInterface)

Private Attributes

- Interface userDefinedInterface

  the interface used for mapping variables to responses

8.90.1 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 layered and nested model extensions. The derived response computation and synchronization functions utilize a single interface to perform the function evaluations.

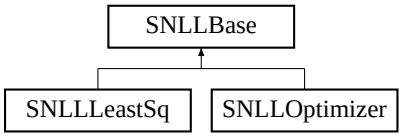
The documentation for this class was generated from the following files:

- SingleModel.H
- SingleModel.C

8.91 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, 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 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,::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, OptppArray< SymmetricMatrix > &hess_g, const size_t &offset)
  convenience function for copying local_fn_hessians to hess_g; used by constraint evaluator functions
- void 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 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, OptimizeClass *the_optimizer, NLP0 *nlf_objective, FDNLF1 *fd_nlf1, FDNLF1 *fd_nlf1_con)

convenience function for setting OPT++ options after the method instantiation

- void pre_run (NLP0 *nlf_objective, NLP *nlp_constraint, const RealVector &init_pt, 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 post_run (NLP0 *nlf_objective)

  convenience function for setting OPT++ options after the method instantiations

Static Protected Member Functions

- void init_fn (int n, 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

- SearchStrategy searchStrat

  enum: LineSearch, TrustRegion, or TrustPDS

- 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

- 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

- bool modeOverrideFlag

  flags OPT++ mode override (for combining value, gradient, and Hessian requests)

- EvalType lastFnEvalLocn

  an enum used to track whether an nlf evaluator or a constraint evaluator was the last location of a function evaluation

- int lastEvalMode

copy of mode from constraint evaluators

- RealVector lastEvalVars

  copy of variables from constraint evaluators

8.91.1 Detailed Description

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

The SNLLBase class provides a common base class for SNLLOptimizer and SNLLLeastSq, both of which are wrappers for OPT++, a C++ optimization library from the Computational Sciences and Mathematics Research (CSMR) department at Sandia's Livermore CA site.

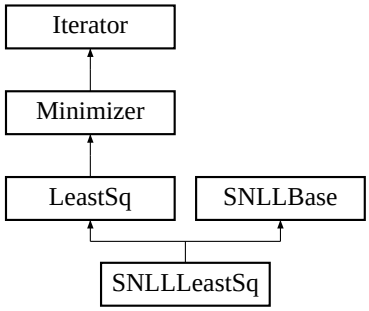
The documentation for this class was generated from the following files:

- SNLLBase.H
- SNLLBase.C

8.92 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.

Static Private Member Functions

- void nlf2_evaluator_gn (int mode, int n, const ColumnVector &x, Real &f, ColumnVector &grad_f, 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.

- void constraint1_evaluator_gn (int mode, int n, const ColumnVector &x, ColumnVector &g,::Matrix &grad_g, int &result_mode)

  constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.

- void constraint2_evaluator_gn (int mode, int n, const ColumnVector &x, ColumnVector &g,::Matrix &grad_g, OptppArray< SymmetricMatrix > &hess_g, int &result_mode)

  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.

Private Attributes

- NLP0 * nlfObjective

  objective NLF base class pointer

- NLP0 * nlfConstraint

  constraint NLF base class pointer

- NLP * nlpConstraint

  constraint NLP pointer

- NLF2 * nlf2

  pointer to objective NLF for full Newton optimizers

- NLF2 * nlf2Con

  pointer to constraint NLF for full Newton optimizers

- NLF1 * nlf1Con

  pointer to constraint NLF for Quasi Newton optimizers

- OptimizeClass * theOptimizer

  optimizer base class pointer

- OptNewton * optnewton

  Newton optimizer pointer.

- OptBCNewton * optbcnewton

  Bound constrained Newton optimizer pointer.

- OptDHNIPS * optdhnips

  Disaggregated Hessian NIPS optimizer ptr.

Static Private Attributes

- SNLLLeastSq * snllLSqInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

8.92.1 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.

8.92.2 Member Function Documentation

8.92.2.1 void nlf2_evaluator_gn (int *mode*, int *n*, const ColumnVector & *x*, Real & *f*, ColumnVector & *grad_f*, 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).

8.92.2.2 void constraint1_evaluator_gn (int *mode*, int *n*, const ColumnVector & *x*, ColumnVector & *g*, ::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.

8.92.2.3 void constraint2_evaluator_gn (int *mode*, int *n*, const ColumnVector & *x*, ColumnVector & *g*, ::Matrix & *grad_g*, OptppArray< 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:

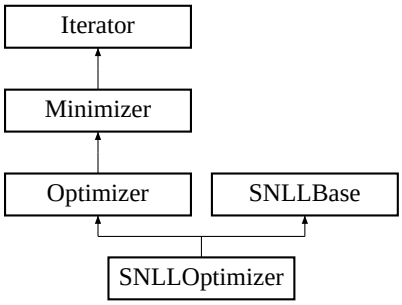
- SNLLLeastSq.H

- SNLLLeastSq.C

8.93 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, int num_lin_ineq, int num_lin_eq, int num_nln_ineq, int num_nln_-eq, 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 ColumnVector &x, Real &f, ColumnVector &grad_f, int &result_mode), void(∗user_con_eval)(int mode, int n, const ColumnVector &x, ColumnVector &g,::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.

Static Private Member Functions

- void nlf0_evaluator (int n, const ColumnVector &x, Real &f, int &result_mode)

  objective function evaluator function for OPT++ methods which require only function values.

- void nlf1_evaluator (int mode, int n, const ColumnVector &x, Real &f, ColumnVector &grad_f, int &result_mode)

  objective function evaluator function which provides function values and gradients to OPT++ methods.

- void nlf2_evaluator (int mode, int n, const ColumnVector &x, Real &f, ColumnVector &grad_f, SymmetricMatrix &hess_f, int &result_mode)

  objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

- void constraint0_evaluator (int n, const ColumnVector &x, ColumnVector &g, int &result_mode)

  constraint evaluator function for OPT++ methods which require only constraint values.

- void constraint1_evaluator (int mode, int n, const ColumnVector &x, ColumnVector &g,::Matrix &grad_g, int &result_mode)

  constraint evaluator function which provides constraint values and gradients to OPT++ methods.

- void constraint2_evaluator (int mode, int n, const ColumnVector &x, ColumnVector &g,::Matrix &grad_g, OptppArray< SymmetricMatrix > &hess_g, int &result_mode)

  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.

Private Attributes

- NLP0 * nlfObjective

  objective NLF base class pointer

- NLP0 * nlfConstraint

  constraint NLF base class pointer

- NLP * nlpConstraint

  constraint NLP pointer

- NLF0 * nlf0

  pointer to objective NLF for nongradient optimizers

- NLF1 * nlf1

  pointer to objective NLF for (analytic) gradient-based optimizers

- NLF1 * nlf1Con

  pointer to constraint NLF for (analytic) gradient-based optimizers

- FDNLF1 * fdnlf1

  pointer to objective NLF for (finite diff) gradient-based optimizers

- FDNLF1 * fdnlf1Con

  pointer to constraint NLF for (finite diff) gradient-based optimizers

- NLF2 * nlf2

  pointer to objective NLF for full Newton optimizers

- NLF2 * nlf2Con

  pointer to constraint NLF for full Newton optimizers

- OptimizeClass * theOptimizer

  optimizer base class pointer

- OptPDS * optpds

  PDS optimizer pointer.

- OptCG * optcg

  CG optimizer pointer.

- OptLBFGS * optlbfgs

  L-BFGS optimizer pointer.

- OptNewton * optnewton

  Newton optimizer pointer.

- OptQNewton * optqnewton

  Quasi-Newton optimizer pointer.

- OptFDNewton * optfdnewton

  Finite Difference Newton optimizer pointer.

- OptBCNewton * optbcnewton

  Bound constrained Newton optimizer pointer.

- OptBCQNewton * optbcqnewton

  Bnd constrained Quasi-Newton optimizer ptr.

- OptBCFDNewton * optbcfdnewton

  Bnd constrained FD-Newton optimizer ptr.

- OptNIPS * optnips

  NIPS optimizer pointer.

- OptQNIPS * optqnips

  Quasi-Newton NIPS optimizer pointer.

- OptFDNIPS * optfdnips

  Finite Difference NIPS optimizer 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

- SNLLOptimizer * snllOptInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

8.93.1 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.

8.93.2 Constructor & Destructor Documentation

8.93.2.1 SNLLOptimizer (Model & *model*)

standard constructor

This constructor is used for normal instantiations using data from the ProblemDescDB.

8.93.2.2 SNLLOptimizer (const RealVector & *initial_point*, const RealVector & *var_lower_bnds*, const RealVector & *var_upper_bnds*, int *num_lin_ineq*, int *num_lin_eq*, int *num_nln_ineq*, int *num_nln_eq*, 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 ColumnVector &x, Real &f, ColumnVector &grad_f, int &result_mode), void(* *user_con_eval*)(int mode, int n, const ColumnVector &x, ColumnVector &g,::Matrix &grad_g, int &result_mode))

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers.

8.93.3 Member Function Documentation

8.93.3.1 void nlf0_evaluator (int *n*, const ColumnVector & *x*, 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).

8.93.3.2 void nlf1_evaluator (int *mode*, int *n*, const ColumnVector & *x*, Real & *f*, 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.

8.93.3.3 void nlf2_evaluator (int *mode*, int *n*, const ColumnVector & *x*, Real & *f*, ColumnVector & *grad_f*, 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.

8.93.3.4 void constraint0_evaluator (int *n*, const ColumnVector & *x*, 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).

8.93.3.5 void constraint1_evaluator (int *mode*, int *n*, const ColumnVector & *x*, ColumnVector & *g*, ::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.

8.93.3.6 void constraint2_evaluator (int *mode*, int *n*, const ColumnVector & *x*, ColumnVector & *g*, ::Matrix & *grad_g*, OptppArray< 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).

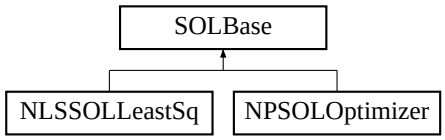
The documentation for this class was generated from the following files:

- SNLLOptimizer.H
- SNLLOptimizer.C

8.94 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_ineq_con, const size_t &num_nln_eq_con, const size_t &num_lin_ineq_con, const size_t &num_lin_eq_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

- 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

- SOLBase * solInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

- 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

8.94.1 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

8.95 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 sortFunction.

Private Attributes

- bool(∗ sortFunction)(const T &, const T &)
  Pointer to test function.

8.95.1 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.

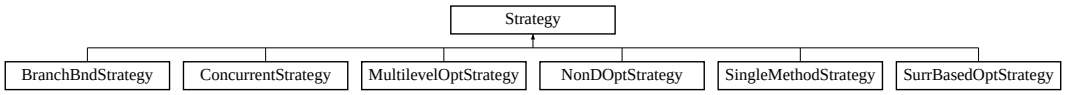
The documentation for this class was generated from the following file:

- DakotaList.H

8.96 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 & strategy_variable_results () const

  return the final strategy solution (variables)

- virtual const Response & strategy_response_results () const

  return the final strategy solution (response)

- virtual IteratorList & iterators (bool recurse_flag=true)

  recurse through nestings/layerings and return all Iterators used in the strategy

- virtual ModelList & models (bool recurse_flag=true)

  recurse through nestings/layerings and return all Models used in the strategy

- 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 (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

- IteratorList iteratorList

  list of iterators returned by iterators()

- ModelList modelList

  list of models returned by models()

Private Member Functions

- Strategy * get_strategy ()

  Used by the envelope to instantiate the correct letter class.

Private Attributes

- Strategy * strategyRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing strategyRep

8.96.1 Detailed Description

Base class for the strategy class hierarchy.

The Strategy class is the base class for the class hierarchy providing the top level control in DAKOTA. The strategy is responsible for creating and managing iterators and models. For memory efficiency and enhanced polymorphism, the strategy hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Strategy) serves as the envelope and one of the derived classes (selected in Strategy::get_strategy()) serves as the letter.

8.96.2 Constructor & Destructor Documentation

8.96.2.1 Strategy ()

default constructor

Default constructor. strategyRep is NULL in this case (a populated problem_db is needed to build a meaningful Strategy object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

8.96.2.2 Strategy (ProblemDescDB & *problem_db*)

envelope constructor

Used in main.C instantiation to build the envelope. This constructor only needs to extract enough data to properly execute get_strategy, since Strategy::Strategy(BaseConstructor, problem_db) builds the actual base class data inherited by the derived strategies.

8.96.2.3 Strategy (const Strategy & *strat*)

copy constructor

Copy constructor manages sharing of strategyRep and incrementing of referenceCount.

8.96.2.4 ~Strategy () [virtual]

destructor

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

8.96.2.5 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).

8.96.3 Member Function Documentation

8.96.3.1 Strategy operator= (const Strategy & *strat*)

assignment operator

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

8.96.3.2 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.

8.96.3.3 void init_communicators (Iterator & *the_iterator*, Model & *the_model*) [protected]

convenience function for allocating comms prior to running an iterator

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator. It does not require a strategyRep forward since it is only used by letter objects.

8.96.3.4 void free_communicators (Model & *the_model*) [protected]

convenience function for deallocating comms after running an iterator

This is a convenience function for encapsulating the deallocation of communicators after running an iterator. It does not require a strategyRep forward since it is only used by letter objects.

8.96.3.5 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.

8.96.3.6 Strategy * get_strategy () [private]

Used by the envelope to instantiate the correct letter class.

Used only by the envelope constructor to initialize strategyRep to the appropriate derived type, as given by the strategyName attribute.

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

- DakotaStrategy.H
- DakotaStrategy.C

8.97 String Class Reference

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

Public Member Functions

- String ()

  Default constructor.

- String (const String &a)

  Default copy constructor.

- String (const char *initial_val)

  Copy constructor from standard C char array.

- ~String ()

  Destructor.

- String & operator= (const String &)

  Normal assignment operator.

- String & operator= (const DAKOTA_BASE_STRING &)

  Assignment operator for base string.

- String & operator= (const char *)

  Assignment operator, standard C char.*

- 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 *subString) const

  Returns true if String contains char substring.*

- char * data () const

  Returns pointer to standard C char array.

8.97.1 Detailed Description

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

The Dakota::String class is the common string class for Dakota. It provides a common interface for string operations whether inheriting from the STL basic_string or the Rogue Wave RWCString class

8.97.2 Member Function Documentation

8.97.2.1 operator const char * () const [inline]

The operator() returns pointer to standard C char array.

The operator () returns a pointer to a char string. Uses the STL c_str() method. This allows for the String to be used in method calls without having to call the data() or c_str() methods.

8.97.2.2 void upper ()

Private method which converts String to upper. Utilizes an STL iterator to step through the string and then calls the STL toupper() method. Needs to be done this way because STL only provides a single char toupper method.

8.97.2.3 void lower ()

Private method which converts String to lower. Utilizes an STL iterator to step through the string and then calls the STL tolower() method. Needs to be done this way because STL only provides a single char tolower method.

8.97.2.4 bool contains (const char * *subString*) const [inline]

Returns true if String contains char* substring.

Returns true of the String contains the char* subString. Calls the STL rfind() method, then checks if substring was found within the String

8.97.2.5 char * data () const [inline]

Returns pointer to standard C char array.

Returns a pointer to C style char array. Needed to mimic the Rogue Wave string class. USE WITH CARE.

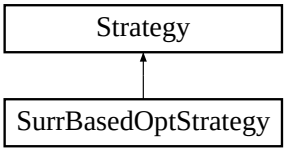
The documentation for this class was generated from the following files:

- DakotaString.H
- DakotaString.C

8.98 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 & strategy_variable_results () const

  return the SBO final solution (variables)

- const Response & strategy_response_results () const

  return the SBO final solution (response)

- IteratorList & iterators (bool recurse_flag=true)

  returns selectedIterator and any subordinate iterators

- ModelList & models (bool recurse_flag=true)

  returns approximateModel and any subordinate models

Private Member Functions

- 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 penalty function near zero)

- void soft_convergence_check (const RealVector &c_vars_center, const RealVector &c_vars_star, const Response &response_center_truth, const Response &response_center_approx, const Response &response_star_truth, const Response &response_star_approx)

check for soft convergence (diminishing returns)

- void compute_penalty (const RealVector &fns_center_truth, const RealVector &fns_star_truth)

  initialize and update the penaltyParameter

- Real compute_penalty_function (const RealVector &fn_vals)

  compute a penalty function from a set of function values

- Real compute_objective (const RealVector &fn_vals)

  compute a single objective value from one or more objective functions

- Real compute_constraint_violation (const RealVector &fn_vals)

  compute the constraint violation from a set of function values

Private Attributes

- Model approximateModel

  the surrogate model (a LayeredModel object)

- Iterator selectedIterator

  the optimizer used on approximateModel

- 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 minTrust-RegionFactor

- 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 (only violated constraints are used in penalty function computations).

- 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)

- Real penaltyParameter

  the penalization factor for violated constraints used in penalty function calculations; increases exponentially with iteration count

- int penaltyIterOffset

  iteration offset used to update the scaling of the penalty parameter

- int sboIterNum

  SBO iteration number.

- int sboIterMax

  maximum number of SBO iterations

- short convergenceFlag

  code indicating satisfaction of hard or soft convergence conditions

- int numFns

  number of response functions

- int 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 within actual_model (e.g., approximateModel = 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

- RealVector multiObjWts

  vector of multiobjective weights.

- RealVector nonlinIneqLowerBnds

  vector of nonlinear inequality constraint lower bounds

- RealVector nonlinIneqUpperBnds

  vector of nonlinear inequality constraint upper bounds

- RealVector nonlinEqTargets

  vector of nonlinear equality constraint targets

- Variables bestVariables

  best variables found in SBO

- Response bestResponses

  best responses found in SBO

8.98.1 Detailed Description

Strategy for provably-convergent surrogate-based optimization.

This strategy uses a LayeredModel to perform optimization based on local, global, or hierarchical surrogates. It achieves provable convergence through the use of a sequence of trust regions and the application of surrogate corrections at the trust region centers.

8.98.2 Member Function Documentation

8.98.2.1 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.

8.98.2.2 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 penalty function near zero)

The hard convergence check computes the 2-norm of the projected gradient of the penalty function (dp/dx = df/dx + 2 r_p g+^T dg+/dx + 2 r_p h+^T dh+/dx) at the trust region center and signals convergence if the 2-norm is close to zero. The projection is needed to remove any gradient component directed into an active bound constraint (since this penalty function does not explicity include Lagrange multipliers times the bound constraints; if it did, the Lagrange multplier for an active bound constraint would zero out the total gradient component).

8.98.2.3 void soft_convergence_check (const RealVector & *c_vars_center*, const RealVector & *c_vars_star*, const Response & *response_center_truth*, const Response & *response_center_approx*, const Response & *response_star_truth*, const Response & *response_star_approx*) [private]

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).

8.98.2.4 void compute_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 iterates which must increase the objective to achieve a reduction in constraint violation. This routine uses the ratio of relative change between center and star points for the objective and constraint violation values to rescale penalty values.

8.98.2.5 Real compute_penalty_function (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.

8.98.2.6 Real compute_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.

8.98.2.7 Real compute_constraint_violation (const RealVector & *fn_vals*) [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 constraintTol allows for a small constraint infeasibility.

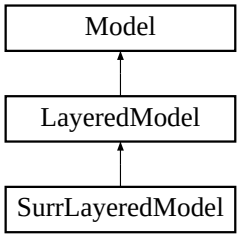
The documentation for this class was generated from the following files:

- SurrBasedOptStrategy.H
- SurrBasedOptStrategy.C

8.99 SurrLayeredModel Class Reference

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

Inheritance diagram for SurrLayeredModel::



Public Member Functions

- SurrLayeredModel (ProblemDescDB &problem_db)
  constructor
- ~SurrLayeredModel ()
  destructor

Protected Member Functions

- void derived_compute_response (const IntArray &asv)
  portion of compute_response() specific to SurrLayeredModel
- void derived_asynch_compute_response (const IntArray &asv)
  portion of asynch_compute_response() specific to SurrLayeredModel
- const ResponseArray & derived_synchronize ()
  portion of synchronize() specific to SurrLayeredModel
- const ResponseList & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to SurrLayeredModel
- const IntList & synchronize_nowait_completions ()
  return completion id's matching response list from derived_synchronize_nowait()
- Model subordinate_model ()
  returns actualModel
- Iterator subordinate_iterator ()
  return daceIterator

- Interface & interface ()

  return approxInterface

- void layering_bypass (bool bypass_flag)

  set layeringBypass flag and pass request on to actualModel for any lower-level layerings.

- void build_approximation ()

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

- void update_approximation (const RealVector &x_star, const Response &response_star)

  Adds a point to a global approximation (request 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

- bool derived_master_overload () const

  prevents overloading the master with a multiprocessor evaluation

- 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 reset_communicators ()

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

- void free_communicators ()

  deallocate communicator partitions for the SurrLayeredModel (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 SurrLayeredModel iteration is complete.

- int total_eval_counter () const

  return the total evaluation count for the SurrLayeredModel (request forwarded to approxInterface)

- int new_eval_counter () const

  return the new evaluation count for the SurrLayeredModel (request forwarded to approxInterface)

Private Member Functions

- void update_actual_model ()

  update actualModel with current variable values/bounds/labels

Private Attributes

- Interface approxInterface

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

- String actualInterfacePointer

  string identifier for the actual interface from the local approximation specification (required for local); used to build actualModel for local approximations

- 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)

- Model actualModel

  the truth model which provides evaluations for building the surrogate (optional for global since restart data may also be used, required for local)

- Iterator daceIterator

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

8.99.1 Detailed Description

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

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

8.99.2 Member Function Documentation

8.99.2.1 void derived_compute_response (const IntArray & *asv*) [protected, virtual]

portion of compute_response() specific to SurrLayeredModel

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

Reimplemented from Model.

8.99.2.2 void derived_asynch_compute_response (const IntArray & *asv*) [protected, virtual]

portion of asynch_compute_response() specific to SurrLayeredModel

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.

8.99.2.3 const ResponseArray & derived_synchronize () [protected, virtual]

portion of synchronize() specific to SurrLayeredModel

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

Reimplemented from Model.

8.99.2.4 const ResponseList & derived_synchronize_nowait () [protected, virtual]

portion of synchronize_nowait() specific to SurrLayeredModel

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

Reimplemented from Model.

8.99.2.5 void build_approximation () [protected, virtual]

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

Build either a global approximation using daceIterator or a local approximation using actualModel. Selection triggers on actualInterfacePointer (required specification for local approximation interfaces, not used in global specification).

Reimplemented from Model.

8.99.2.6 bool derived_master_overload () const [inline, protected, virtual]

prevents overloading the master with a multiprocessor evaluation

compute_response calls never overload the master since there is no parallelism in the use of approxInterface. Derived master overload for actualModel is handled separately in actualModel.compute_response() (within daceIterator.run_iterator(), etc.).

Reimplemented from Model.

8.99.2.7 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.

8.99.2.8 void update_actual_model () [private]

update actualModel with current variable values/bounds/labels

Update variables data within actualModel using values and labels from currentVariables and bounds from userDefinedVarConstraints.

8.99.3 Member Data Documentation

8.99.3.1 String actualInterfacePointer [private]

string identifier for the actual interface from the local approximation specification (required for local); used to build actualModel for local approximations

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

8.99.3.2 Model actualModel [private]

the truth model which provides evaluations for building the surrogate (optional for global since restart data may also be used, required for local)

There are no restrictions on actualModel in the global case, so arbitrary nestings are possible. In the local case, model_type must be set to "single" to avoid recursion on SurrLayeredModel, since there is no additional method specification.

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

- SurrLayeredModel.H
- SurrLayeredModel.C

8.100 SurrogateDataPoint Class Reference

Simple 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

- int operator== (const SurrogateDataPoint &sdp) const

  equality operator

Public Attributes

- RealVector continuousVars

  continuous variables

- Real responseFn

  truth response function value

- RealBaseVector responseGrad

  truth response function gradient

- RealMatrix responseHess

  truth response function Hessian

8.100.1 Detailed Description

Simple 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. Data is public to avoid maintaining set/get functions, but is still encapsulated within Approximation since Approximation::currentPoints is protected (a similar model is used with with Data class objects contained in ProblemDescDB).

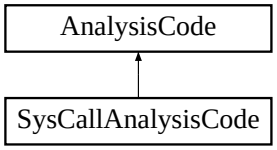
The documentation for this class was generated from the following file:

- DakotaApproximation.H

8.101 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)

8.101.1 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.

8.101.2 Member Function Documentation

8.101.2.1 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.

8.101.2.2 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.

8.101.2.3 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.

8.101.2.4 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.

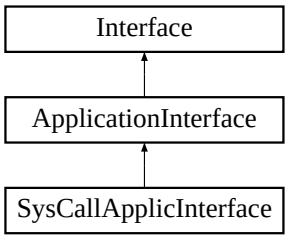
The documentation for this class was generated from the following files:

- SysCallAnalysisCode.H
- SysCallAnalysisCode.C

8.102 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, const size_t &num_fns)

  constructor

- ~SysCallApplicInterface ()

  destructor

- void derived_map (const Variables &vars, const IntArray &asv, 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)

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

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.

- IntList sysCallList

  list of function evaluation id's for active asynchronous system call evaluations

- IntList failIdList

  list of function evaluation id's for tracking response file read failures

- IntList failCountList

  list containing the number of response read failures for each function evaluation identified in failIdList

8.102.1 Detailed Description

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

SysCallApplicInterface uses a SysCallAnalysisCode object for performing simulation invocations.

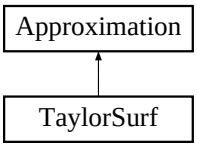
The documentation for this class was generated from the following files:

- SysCallApplicInterface.H
- SysCallApplicInterface.C

8.103 TaylorSurf Class Reference

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

Inheritance diagram for TaylorSurf::



Public Member Functions

- TaylorSurf (const ProblemDescDB &problem_db, const size_t &num_acv)

  constructor

- ~TaylorSurf ()

  destructor

Protected Member Functions

- void find_coefficients ()

  calculate the data fit coefficients using the currentPoints list of SurrogateDataPoints

- int required_samples ()

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

- 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

Private Attributes

- bool secondOrderFlag

  flag to indicate a 2nd-order Taylor series with a Hessian term

8.103.1 Detailed Description

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

The TaylorSurf 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.

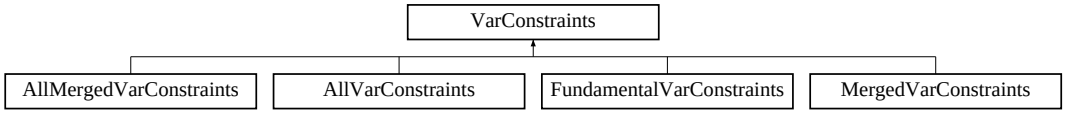
The documentation for this class was generated from the following files:

- TaylorSurf.H
- TaylorSurf.C

8.104 VarConstraints Class Reference

Base class for the variable constraints class hierarchy.

Inheritance diagram for VarConstraints::



Public Member Functions

- VarConstraints ()
  default constructor
- VarConstraints (const ProblemDescDB &problem_db, const String &vars_type)
  standard constructor
- VarConstraints (const VarConstraints &vc)
  copy constructor
- virtual ~VarConstraints ()
  destructor
- VarConstraints operator= (const VarConstraints &vc)
  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

- const RealVector & linear_ineq_constraint_lower_bounds () const

  return the linear inequality constraint lower bounds

- const RealVector & linear_ineq_constraint_upper_bounds () const

  return the linear inequality constraint upper bounds

- const RealMatrix & linear_eq_constraint_coeffs () const

  return the linear equality constraint coefficients

- const RealVector & linear_eq_constraint_targets () const

  return the linear equality constraint targets

Protected Member Functions

- VarConstraints (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 manage_linear_constraints (const ProblemDescDB &problem_db)

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

Protected Attributes

- String variablesType

  All, Merged, AllMerged, or Fundamental.

- size_t numLinearIneqConstraints

  number of linear inequality constraints

- size_t numLinearEqConstraints

  number of linear equality constraints

- RealMatrix linearIneqConstraintCoeffs

  linear inequality constraint coefficients

- RealMatrix linearEqConstraintCoeffs

  linear equality constraint coefficients

- RealVector linearIneqConstraintLowerBnds

  linear inequality constraint lower bounds

- RealVector linearIneqConstraintUpperBnds

  linear inequality constraint upper bounds

- RealVector linearEqConstraintTargets

  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

- VarConstraints * get_var_constraints (const ProblemDescDB &problem_db)

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

Private Attributes

- VarConstraints * varConstraintsRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing varConstraintsRep

8.104.1 Detailed Description

Base class for the variable constraints class hierarchy.

The VarConstraints class is the base class for the class hierarchy managing linear and bound constraints on the variables. Using the variable lower and upper bounds arrays and linear constraint coefficients and bounds from the input specification, different derived classes define different views of this data. 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 (VarConstraints) serves as the envelope and one of the derived classes (selected in VarConstraints::get_var_constraints()) serves as the letter.

8.104.2 Constructor & Destructor Documentation

8.104.2.1 VarConstraints ()

default constructor

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

8.104.2.2 VarConstraints (const ProblemDescDB & *problem_db*, const String & *vars_type*)

standard constructor

The envelope constructor only needs to extract enough data to properly execute get_var_constraints, since the constructor overloaded with BaseConstructor builds the actual base class data inherited by the derived classes.

8.104.2.3 VarConstraints (const VarConstraints & *vc*)

copy constructor

Copy constructor manages sharing of varConstraintsRep and incrementing of referenceCount.

8.104.2.4 ~VarConstraints () [virtual]

destructor

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

8.104.2.5 VarConstraints (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 derived classes. get_var_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_var_constraints() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~VarConstraints).

8.104.3 Member Function Documentation

8.104.3.1 VarConstraints operator= (const VarConstraints & *vc*)

assignment operator

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

8.104.3.2 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.

8.104.3.3 VarConstraints * get_var_constraints (const ProblemDescDB & *problem_db*) [private]

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

Initializes varConstraintsRep to the appropriate derived type, as given by the variablesType attribute.

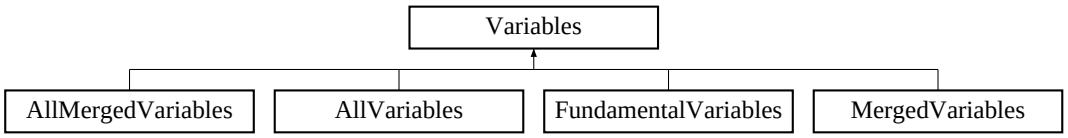
The documentation for this class was generated from the following files:

- DakotaVarConstraints.H
- DakotaVarConstraints.C

8.105 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 String &vars_type)
  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 size_t cv () const
  Returns number of active continuous vars.
- virtual size_t dv () const
  Returns number of active discrete 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 IntVector all_discrete_variables () const
  returns a single array with all discrete variables
- 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
- 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 String & variables_type () const

returns the variables type: All, Merged, AllMerged, or Fundamental

- const StringArray & continuous_variable_types () const
  return the active continuous variable types

- const StringArray & discrete_variable_types () const
  return the active discrete variable types

Protected Member Functions

- Variables (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)

Protected Attributes

- IntList mergedIntegerList
  the list of discrete variables for which integrality is relaxed by merging them into a continuous array

- String variablesType
  All, Merged, AllMerged, or Fundamental.

- StringArray continuousVarTypes
  array of variable types for the active continuous variables

- StringArray discreteVarTypes
  array of variable types for the active discrete variables

- 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 String &vars_type) const
  Used by the alternate envelope constructor, by read functions, and by copy() to instantiate a new letter class.

Private Attributes

- Variables * variablesRep

  pointer to the letter (initialized only for the envelope)

- int referenceCount

  number of objects sharing variablesRep

Friends

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

  equality operator

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

  inequality operator

8.105.1 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.

8.105.2 Constructor & Destructor Documentation

8.105.2.1 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.

8.105.2.2 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.

8.105.2.3 Variables (const String & *vars_type*)

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(vars_type), which invokes the default constructor of the derived letter class, which in turn invokes the default constructor of the base class.

8.105.2.4 Variables (const Variables & *vars*)

copy constructor

Copy constructor manages sharing of variablesRep and incrementing of referenceCount.

8.105.2.5 ~Variables () [virtual]

destructor

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

8.105.2.6 Variables (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 derived classes. get_variables() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_variables() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Variables).

8.105.3 Member Function Documentation

8.105.3.1 Variables operator= (const Variables & *vars*)

assignment operator

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

8.105.3.2 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).

8.105.3.3 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.

8.105.3.4 Variables * get_variables (const String & *vars_type*) 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 the vars_type attribute. The default derived class constructors are invoked.

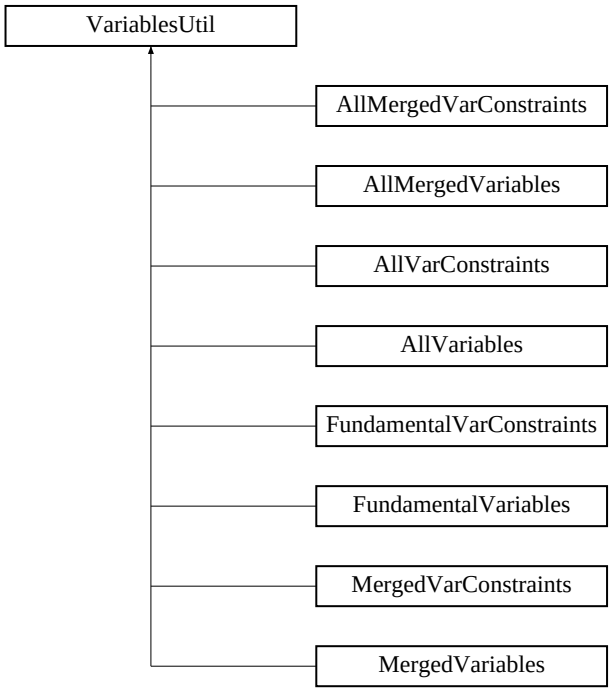
The documentation for this class was generated from the following files:

- DakotaVariables.H
- DakotaVariables.C

8.106 VariablesUtil Class Reference

Utility class for the Variables and VarConstraints 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)

  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_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)

8.106.1 Detailed Description

Utility class for the Variables and VarConstraints 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 VarConstraints hierarchies use multiple inheritance to inherit these utilities.

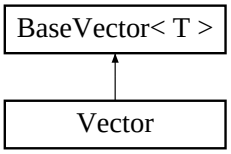
The documentation for this class was generated from the following file:

- VariablesUtil.H

8.107 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 print (ostream &s) const

  Prints a Vector to an output stream.

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

  Prints a Vector and associated label array to an output stream.

- void print_partial (ostream &s, size_t start_index, size_t num_items) const

  Prints part of a Vector to an output stream.

- void print_partial (ostream &s, size_t start_index, size_t num_items, const Array< String > &label_array) const

  Prints part of a Vector and the corresponding labels to an output stream.

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

  Prints a Vector and associated label array to an output stream in aprepro format.

- void print_partial_aprepro (ostream &s, size_t start_index, size_t num_items, const Array< String > &label_array) const

  Prints part of a Vector and the corresponding labels to an output stream in aprepro format.

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

  Prints a Vector and associated label array in annotated form to an output stream.

- void print_tabular (ostream &s) const

  Prints a Vector in tabular form to an output stream.

- void print_partial_tabular (ostream &s, size_t start_index, size_t num_items) const

  Prints 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 print (BoStream &s, const Array< String > &label_array) const

  Prints 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 print (MPIPackBuffer &s) const

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

- void print (MPIPackBuffer &s, const Array< String > &label_array) const

  Writes a Vector and associated label array to a buffer prior to an MPI send.

8.107.1 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/print vectors in a variety of ways

8.107.2 Constructor & Destructor Documentation

8.107.2.1 Vector (const T * *p*, size_t *len*) [inline]

Constructor which copies len entries from T*.

Assigns size values from p into array.

8.107.3 Member Function Documentation

8.107.3.1 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 9

DAKOTA File Documentation

9.1 keywordtable.C File Reference

file containing keywords for the strategy, method, 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.

9.1.1 Detailed Description

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

9.2 main.C File Reference

file containing the main program for DAKOTA

Functions

- int main (int argc, char *argv[ ])
  The main DAKOTA program.

Variables

- int write_precision = 10
  used in ostream data output functions

9.2.1 Detailed Description

file containing the main program for DAKOTA

9.2.2 Function Documentation

9.2.2.1 int main (int *argc*, char * *argv*[ ])

The main DAKOTA program.

Manage command line inputs, input files, restart file(s), output streams, and top level parallel iterator communicators. Instantiate the Strategy and invoke its run_strategy() virtual function.

9.3 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.

Variables

- int write_precision = 16

  used in ostream data output functions

9.3.1 Detailed Description

file containing the DAKOTA restart utility main program

9.3.2 Function Documentation

9.3.2.1 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.

9.3.2.2 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).

9.3.2.3 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.

9.3.2.4 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).

9.3.2.5 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.

9.3.2.6 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 10

Interfacing with DAKOTA as a Library

10.1 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, 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. 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; man-

aging 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 modifying 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.

10.2 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).

10.3 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, 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, 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, so it is not necessary to instantiate and publish a DataStrategy object if coordination of multiple iterators and models is 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.

10.4 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.

10.5 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.

10.5.1 Extension

The first approach involves extending the existing DirectFnApplicInterface class to support additional direct simulation interfaces. In this case, a new interface member function can be added to Dakota/src/DirectFnApplicInterface.[CH] for the simulation of interest using the prototype:

```
int sim(const Variables& vars, const IntArray& asv, Response& response);
```

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.

10.5.2 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  = selected_strategy.models();
Model&     first_model = *all_models.begin();
Interface& interface   = first_model.interface();
// plug in the new direct interface instance
interface.assign_rep(new DirectFnApplicInterface(problem_db, num_fns));
// 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 = selected_strategy.models();
// 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() == "application_direct") {
    // plug in the new direct interface instance
    interface.assign_rep(new DirectFnApplicInterface(problem_db, num_fns));
    // 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), ApplicationInterface::analysisDrivers provides the analysis driver names specified by the user in the input file, and ApplicationInterface::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.

10.6 Executing the strategy

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

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

10.7 Retrieving data after a run

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

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

// retrieve the final response values
const Response& resp  = selected_strategy.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.

10.8 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 11

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.

11.1 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.

11.2 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.

11.3 Analyses within each function evaluation

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

Chapter 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 arose from the desire to have member functions which set and return the value of a 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 change
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., Dakota-Response.[CH] files contain Dakota::Response and Dakota::ResponseRep classes, and DakotaBin-Stream.[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 objec
//- Owner:       Mike Eldred
//- Version: $Id: RecommendPract.dox,v 1.7 2004/05/22 00:29:02 mseldre Exp $
```

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).
- 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 ProblemDescDB 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 ProblemDescDB.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 ProblemDescDB.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/ProblemDescDB.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 ProblemDescDB.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 ProblemDescDB.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 ProblemDescDB::idr_get_int_table(), ProblemDescDB::idr_get_real_table(), and ProblemDescDB::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 ProblemDescDB.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::methodList, ProblemDescDB::variablesList, ProblemDescDB::interfaceList, ProblemDescDB::responsesList, 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 ProblemDescDB.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).

13.4.3 Augment/update get_<data_type>() functions

The final update step for ProblemDescDB.C involves extending the database retrieval functions. 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 DakotaString& 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 (*variablesIter).continuousDesignVars;
```

appears at the top of ProblemDescDB::get_drv(). Based on the identifier string, it returns the `continuousDesignVars` attribute from a DataVariables object. Since there may be multiple variables specifications, the `variablesIter` list iterator identifies which node in the list of DataVariables objects is used. In particular, `variablesList` 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 `variablesIter`, which is set in a `set_db_list_nodes()` operation that will not be described here.

There may be multiple DataVariables, DataInterface, DataResponses, and/or DataMethod 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 (∗listIter).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 (*methodIter).methodSetting;
```

A strategy specification addition would not use a (∗listIter) syntax, but would instead look like:

```
else if (entry_name == "strategy.strategy_name.strategy_setting")
  return strategySpec.strategySetting;
```

13.5 Update Corresponding Data Classes

In this step, we extend the Data class definitions (DataStrategy, DataMethod, DataVariables, DataInterface, and/or DataResponses) to include the new attributes referenced in Update keyword handler logic blocks and Augment/update get_<data_type>() functions.

13.5.1 Update the Data class header file

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

13.5.2 Update the .C file

Define defaults for the new attributes in the constructor initialization list (or in the case of DataMethod, in the body of the constructor for readability). 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 attention to using a consistent ordering.

13.6 Use get_<data_type>() Functions

At this point, the new specifications have been mapped through all of the database classes. The only remaining step is to retrieve the new data within the constructors of the classes that need it. This is done

by invoking the get_<data_type>() function on the ProblemDescDB object using the identifier string you selected in Augment/update get_<data_type>() functions. For example, from DakotaModel.C:

```
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., `methodIter`, `interfaceIter`, `variablesIter`, and `responsesIter`) 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.

13.7 Update the Documentation

Doxygen comments should be added to the Data class headers for the new attributes, and the reference manual sections describing the portions of **dakota.input.spec** that have been modified should be updated.

Index