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

Version 5.0 Developers Manual

Brian M. Adams, Keith R. Dalbey, Michael S. Eldred, David M. Gay, Laura P. Swiler
Optimization and Uncertainty Quantification Department

William J. Bohnhoff
Radiation Transport Department

John P. Eddy
System Readiness and Sustainment Technologies Department

Karen Haskell
Scientific Applications and User Support Department

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

Patricia D. Hough
Informatics and Decision Sciences Department

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

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

This report serves as a developers manual for the DAKOTA software and describes the DAKOTA class hierarchies and their interrelationships. It derives directly from annotation of the actual source code and provides detailed class documentation, including all member functions and attributes.
# Contents

1 DAKOTA Developers Manual .................................................. 7
   1.1 Introduction ...................................................................... 7
   1.2 Overview of DAKOTA ................................................... 7
   1.3 Services ........................................................................... 12
   1.4 Additional Resources ................................................... 13

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

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

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

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

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

7 DAKOTA Namespace Documentation ..................................... 29
   7.1 Dakota Namespace Reference ...................................... 29
   7.2 SIM Namespace Reference ........................................... 154

8 DAKOTA Class Documentation ............................................. 155
   8.1 ActiveSet Class Reference .......................................... 155
   8.2 AnalysisCode Class Reference ...................................... 158
<table>
<thead>
<tr>
<th>Section</th>
<th>Class Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3</td>
<td>Analyzer Class Reference</td>
<td>163</td>
</tr>
<tr>
<td>8.4</td>
<td>ApplicationInterface Class Reference</td>
<td>167</td>
</tr>
<tr>
<td>8.5</td>
<td>Approximation Class Reference</td>
<td>178</td>
</tr>
<tr>
<td>8.6</td>
<td>ApproximationInterface Class Reference</td>
<td>185</td>
</tr>
<tr>
<td>8.7</td>
<td>APPSEvalMgr Class Reference</td>
<td>189</td>
</tr>
<tr>
<td>8.8</td>
<td>APPSOptimizer Class Reference</td>
<td>192</td>
</tr>
<tr>
<td>8.9</td>
<td>Array Class Template Reference</td>
<td>195</td>
</tr>
<tr>
<td>8.10</td>
<td>BaseConstructor Struct Reference</td>
<td>199</td>
</tr>
<tr>
<td>8.11</td>
<td>BasisPolyApproximation Class Reference</td>
<td>200</td>
</tr>
<tr>
<td>8.12</td>
<td>BasisPolynomial Class Reference</td>
<td>205</td>
</tr>
<tr>
<td>8.13</td>
<td>BiStream Class Reference</td>
<td>212</td>
</tr>
<tr>
<td>8.14</td>
<td>BoStream Class Reference</td>
<td>215</td>
</tr>
<tr>
<td>8.15</td>
<td>COLINApplication Class Reference</td>
<td>218</td>
</tr>
<tr>
<td>8.16</td>
<td>COLINOptimizer Class Template Reference</td>
<td>221</td>
</tr>
<tr>
<td>8.17</td>
<td>ColinPoint Class Reference</td>
<td>225</td>
</tr>
<tr>
<td>8.18</td>
<td>CollaborativeHybridStrategy Class Reference</td>
<td>226</td>
</tr>
<tr>
<td>8.19</td>
<td>CommandLineHandler Class Reference</td>
<td>228</td>
</tr>
<tr>
<td>8.20</td>
<td>CommandShell Class Reference</td>
<td>230</td>
</tr>
<tr>
<td>8.21</td>
<td>ConcurrentStrategy Class Reference</td>
<td>232</td>
</tr>
<tr>
<td>8.22</td>
<td>CONMINOptimizer Class Reference</td>
<td>235</td>
</tr>
<tr>
<td>8.23</td>
<td>Constraints Class Reference</td>
<td>243</td>
</tr>
<tr>
<td>8.24</td>
<td>CtelRegexp Class Reference</td>
<td>255</td>
</tr>
<tr>
<td>8.25</td>
<td>DataFitSurrModel Class Reference</td>
<td>258</td>
</tr>
<tr>
<td>8.26</td>
<td>DataInterface Class Reference</td>
<td>267</td>
</tr>
<tr>
<td>8.27</td>
<td>DataMethod Class Reference</td>
<td>268</td>
</tr>
<tr>
<td>8.28</td>
<td>DataMethodRep Class Reference</td>
<td>269</td>
</tr>
<tr>
<td>8.29</td>
<td>DataModel Class Reference</td>
<td>281</td>
</tr>
<tr>
<td>8.30</td>
<td>DataModelRep Class Reference</td>
<td>282</td>
</tr>
<tr>
<td>8.31</td>
<td>DataResponses Class Reference</td>
<td>286</td>
</tr>
<tr>
<td>8.32</td>
<td>DataResponsesRep Class Reference</td>
<td>287</td>
</tr>
<tr>
<td>8.33</td>
<td>DataStrategy Class Reference</td>
<td>291</td>
</tr>
<tr>
<td>8.34</td>
<td>DataStrategyRep Class Reference</td>
<td>292</td>
</tr>
<tr>
<td>8.35</td>
<td>DataVariables Class Reference</td>
<td>295</td>
</tr>
<tr>
<td>8.36</td>
<td>DataVariablesRep Class Reference</td>
<td>297</td>
</tr>
<tr>
<td>Section</td>
<td>Class/Struct Reference</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.37</td>
<td>DDACEDesignCompExp Class Reference</td>
<td>307</td>
</tr>
<tr>
<td>8.38</td>
<td>DirectApplicInterface Class Reference</td>
<td>311</td>
</tr>
<tr>
<td>8.39</td>
<td>DOTOptimizer Class Reference</td>
<td>318</td>
</tr>
<tr>
<td>8.40</td>
<td>EffGlobalMinimizer Class Reference</td>
<td>323</td>
</tr>
<tr>
<td>8.41</td>
<td>EmbeddedHybridStrategy Class Reference</td>
<td>326</td>
</tr>
<tr>
<td>8.42</td>
<td>ErrorTable Struct Reference</td>
<td>328</td>
</tr>
<tr>
<td>8.43</td>
<td>ForkAnalysisCode Class Reference</td>
<td>329</td>
</tr>
<tr>
<td>8.44</td>
<td>ForkApplicInterface Class Reference</td>
<td>331</td>
</tr>
<tr>
<td>8.45</td>
<td>FSUDesignCompExp Class Reference</td>
<td>334</td>
</tr>
<tr>
<td>8.46</td>
<td>FunctionCompare Class Template Reference</td>
<td>338</td>
</tr>
<tr>
<td>8.47</td>
<td>GaussProcApproximation Class Reference</td>
<td>339</td>
</tr>
<tr>
<td>8.48</td>
<td>GenLaguerreOrthogPolynomial Class Reference</td>
<td>345</td>
</tr>
<tr>
<td>8.49</td>
<td>GetLongOpt Class Reference</td>
<td>347</td>
</tr>
<tr>
<td>8.50</td>
<td>Graphics Class Reference</td>
<td>351</td>
</tr>
<tr>
<td>8.51</td>
<td>GridApplicInterface Class Reference</td>
<td>355</td>
</tr>
<tr>
<td>8.52</td>
<td>HermiteOrthogPolynomial Class Reference</td>
<td>358</td>
</tr>
<tr>
<td>8.53</td>
<td>HierarchSurrModel Class Reference</td>
<td>360</td>
</tr>
<tr>
<td>8.54</td>
<td>HybridStrategy Class Reference</td>
<td>364</td>
</tr>
<tr>
<td>8.55</td>
<td>Interface Class Reference</td>
<td>366</td>
</tr>
<tr>
<td>8.56</td>
<td>InterpPolyApproximation Class Reference</td>
<td>375</td>
</tr>
<tr>
<td>8.57</td>
<td>Iterator Class Reference</td>
<td>382</td>
</tr>
<tr>
<td>8.58</td>
<td>JacobiOrthogPolynomial Class Reference</td>
<td>394</td>
</tr>
<tr>
<td>8.59</td>
<td>JEGAOptimizer Class Reference</td>
<td>396</td>
</tr>
<tr>
<td>8.60</td>
<td>JEGAOptimizer::Driver Class Reference</td>
<td>404</td>
</tr>
<tr>
<td>8.61</td>
<td>JEGAOptimizer::Evaluator Class Reference</td>
<td>406</td>
</tr>
<tr>
<td>8.62</td>
<td>JEGAOptimizer::EvaluatorCreator Class Reference</td>
<td>412</td>
</tr>
<tr>
<td>8.63</td>
<td>LagrangeInterpPolynomial Class Reference</td>
<td>414</td>
</tr>
<tr>
<td>8.64</td>
<td>LaguerreOrthogPolynomial Class Reference</td>
<td>416</td>
</tr>
<tr>
<td>8.65</td>
<td>LeastSq Class Reference</td>
<td>418</td>
</tr>
<tr>
<td>8.66</td>
<td>LegendreOrthogPolynomial Class Reference</td>
<td>422</td>
</tr>
<tr>
<td>8.67</td>
<td>List Class Template Reference</td>
<td>424</td>
</tr>
<tr>
<td>8.68</td>
<td>MergedConstraints Class Reference</td>
<td>428</td>
</tr>
<tr>
<td>8.69</td>
<td>MergedVariables Class Reference</td>
<td>430</td>
</tr>
<tr>
<td>8.70</td>
<td>Minimizer Class Reference</td>
<td>433</td>
</tr>
<tr>
<td>Section</td>
<td>Class Reference</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.71</td>
<td>MixedConstraints Class Reference</td>
<td>441</td>
</tr>
<tr>
<td>8.72</td>
<td>MixedVariables Class Reference</td>
<td>443</td>
</tr>
<tr>
<td>8.73</td>
<td>Model Class Reference</td>
<td>445</td>
</tr>
<tr>
<td>8.74</td>
<td>Model::FDhelp Struct Reference</td>
<td>484</td>
</tr>
<tr>
<td>8.75</td>
<td>MPIPackBuffer Class Reference</td>
<td>485</td>
</tr>
<tr>
<td>8.76</td>
<td>MPIUnpackBuffer Class Reference</td>
<td>488</td>
</tr>
<tr>
<td>8.77</td>
<td>NCSUOptimizer Class Reference</td>
<td>491</td>
</tr>
<tr>
<td>8.78</td>
<td>NestedModel Class Reference</td>
<td>494</td>
</tr>
<tr>
<td>8.79</td>
<td>NIDRProblemDescDB Class Reference</td>
<td>501</td>
</tr>
<tr>
<td>8.80</td>
<td>NL2Res Struct Reference</td>
<td>506</td>
</tr>
<tr>
<td>8.81</td>
<td>NL2SOLLestSq Class Reference</td>
<td>507</td>
</tr>
<tr>
<td>8.82</td>
<td>NLQLPOptimizer Class Reference</td>
<td>510</td>
</tr>
<tr>
<td>8.83</td>
<td>NLSSOLLestSq Class Reference</td>
<td>515</td>
</tr>
<tr>
<td>8.84</td>
<td>NoDBBaseConstructor Struct Reference</td>
<td>517</td>
</tr>
<tr>
<td>8.85</td>
<td>NonD Class Reference</td>
<td>518</td>
</tr>
<tr>
<td>8.86</td>
<td>NonDAdaptImpSampling Class Reference</td>
<td>526</td>
</tr>
<tr>
<td>8.87</td>
<td>NonDBayesCal Class Reference</td>
<td>530</td>
</tr>
<tr>
<td>8.88</td>
<td>NonDExpansion Class Reference</td>
<td>532</td>
</tr>
<tr>
<td>8.89</td>
<td>NonDGlobalEvidence Class Reference</td>
<td>535</td>
</tr>
<tr>
<td>8.90</td>
<td>NonDGlobalInterval Class Reference</td>
<td>537</td>
</tr>
<tr>
<td>8.91</td>
<td>NonDGlobalReliability Class Reference</td>
<td>541</td>
</tr>
<tr>
<td>8.92</td>
<td>NonDGlobalSingleInterval Class Reference</td>
<td>544</td>
</tr>
<tr>
<td>8.93</td>
<td>NonDIncrementLHSampling Class Reference</td>
<td>546</td>
</tr>
<tr>
<td>8.94</td>
<td>NonDIntegration Class Reference</td>
<td>548</td>
</tr>
<tr>
<td>8.95</td>
<td>NonDInterval Class Reference</td>
<td>551</td>
</tr>
<tr>
<td>8.96</td>
<td>NonDLHSEvidence Class Reference</td>
<td>554</td>
</tr>
<tr>
<td>8.97</td>
<td>NonDLHSInterval Class Reference</td>
<td>556</td>
</tr>
<tr>
<td>8.98</td>
<td>NonDLHSSampling Class Reference</td>
<td>558</td>
</tr>
<tr>
<td>8.99</td>
<td>NonDLHSSingleInterval Class Reference</td>
<td>561</td>
</tr>
<tr>
<td>8.100</td>
<td>NonDLocalEvidence Class Reference</td>
<td>563</td>
</tr>
<tr>
<td>8.101</td>
<td>NonDLocalInterval Class Reference</td>
<td>565</td>
</tr>
<tr>
<td>8.102</td>
<td>NonDLocalReliability Class Reference</td>
<td>568</td>
</tr>
<tr>
<td>8.103</td>
<td>NonDLocalSingleInterval Class Reference</td>
<td>574</td>
</tr>
<tr>
<td>8.104</td>
<td>NonDPolynomialChaos Class Reference</td>
<td>576</td>
</tr>
<tr>
<td>Section Number</td>
<td>Class Reference</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.105</td>
<td>NonDQuadrature</td>
<td>578</td>
</tr>
<tr>
<td>8.106</td>
<td>NonDReliability</td>
<td>581</td>
</tr>
<tr>
<td>8.107</td>
<td>NonDSampling</td>
<td>585</td>
</tr>
<tr>
<td>8.108</td>
<td>NonDSparseGrid</td>
<td>590</td>
</tr>
<tr>
<td>8.109</td>
<td>NonDStochCollocation</td>
<td>596</td>
</tr>
<tr>
<td>8.110</td>
<td>NPSOLOptimizer</td>
<td>597</td>
</tr>
<tr>
<td>8.111</td>
<td>NumericGenOrthogonalPolynomial</td>
<td>601</td>
</tr>
<tr>
<td>8.112</td>
<td>Optimizer</td>
<td>606</td>
</tr>
<tr>
<td>8.113</td>
<td>OrthogonalPolynomial</td>
<td>611</td>
</tr>
<tr>
<td>8.114</td>
<td>OrthogPolyApproximation</td>
<td>613</td>
</tr>
<tr>
<td>8.115</td>
<td>ParallelConfiguration</td>
<td>621</td>
</tr>
<tr>
<td>8.116</td>
<td>ParallelDirectAplicInterface</td>
<td>623</td>
</tr>
<tr>
<td>8.117</td>
<td>ParallelLevel</td>
<td>624</td>
</tr>
<tr>
<td>8.118</td>
<td>ParallelLibrary</td>
<td>627</td>
</tr>
<tr>
<td>8.119</td>
<td>ParamResponsePair</td>
<td>640</td>
</tr>
<tr>
<td>8.120</td>
<td>ParamStudy</td>
<td>644</td>
</tr>
<tr>
<td>8.121</td>
<td>partial_prp_equality</td>
<td>649</td>
</tr>
<tr>
<td>8.122</td>
<td>partial_prp_hash</td>
<td>650</td>
</tr>
<tr>
<td>8.123</td>
<td>ProblemDescDB</td>
<td>651</td>
</tr>
<tr>
<td>8.124</td>
<td>PStudyDACE</td>
<td>661</td>
</tr>
<tr>
<td>8.125</td>
<td>PSUADEDesignCompExp</td>
<td>664</td>
</tr>
<tr>
<td>8.126</td>
<td>RecastBaseConstructor</td>
<td>667</td>
</tr>
<tr>
<td>8.127</td>
<td>RecastModel</td>
<td>668</td>
</tr>
<tr>
<td>8.128</td>
<td>Response</td>
<td>674</td>
</tr>
<tr>
<td>8.129</td>
<td>ResponseRep</td>
<td>679</td>
</tr>
<tr>
<td>8.130</td>
<td>SensAnalysisGlobal</td>
<td>685</td>
</tr>
<tr>
<td>8.131</td>
<td>SequentialHybridStrategy</td>
<td>687</td>
</tr>
<tr>
<td>8.132</td>
<td>SerialDirectAplicInterface</td>
<td>692</td>
</tr>
<tr>
<td>8.133</td>
<td>SingleMethodStrategy</td>
<td>693</td>
</tr>
<tr>
<td>8.134</td>
<td>SingleModel</td>
<td>695</td>
</tr>
<tr>
<td>8.135</td>
<td>SNLLBase</td>
<td>698</td>
</tr>
<tr>
<td>8.136</td>
<td>SNLLLeastSq</td>
<td>701</td>
</tr>
<tr>
<td>8.137</td>
<td>SNLLOptimizer</td>
<td>705</td>
</tr>
<tr>
<td>8.138</td>
<td>SOLBase</td>
<td>711</td>
</tr>
</tbody>
</table>
11.4 Update ProblemDescDB.C in Dakota/src

11.5 Update Corresponding Data Classes

11.6 Use get_<data_type>() Functions

11.7 Update the Documentation

12 Interfacing with DAKOTA as a Library

12.1 Introduction

12.2 Quick start: examples and test code

12.3 Comparison to main.C

12.4 Problem database population

12.5 Instantiating the strategy

12.6 Defining the direct application interface

12.7 Additional updates

12.8 Executing the strategy

12.9 Retrieving data after a run

12.10 Linking against the DAKOTA library

12.11 Summary

13 Performing Function Evaluations

13.1 Synchronous function evaluations

13.2 Asynchronous function evaluations

13.3 Analyses within each function evaluation

14 Software Tools for DAKOTA Development

14.1 Introduction

14.2 Subversion for Version Control

14.3 GNU Autotools for Configuration Management
Chapter 1

DAKOTA Developers Manual

Author:
Brian M. Adams, William J. Bohnhoff, Keith R. Dalbey, John P. Eddy, Michael S. Eldred, David M. Gay, Karen Haskell, Patricia D. Hough, Laura P. Swiler

1.1 Introduction

The DAKOTA (Design Analysis Kit for Optimization and Terascale Applications) toolkit provides a flexible, extensible interface between analysis codes and iteration methods. DAKOTA contains algorithms for optimization with gradient and nongradient-based methods, uncertainty quantification with sampling, reliability, stochastic expansion, and interval estimation methods, parameter estimation with nonlinear least squares methods, and sensitivity/variance analysis with design of experiments and parameter study capabilities. These capabilities may be used on their own or as components within advanced algorithms such as surrogate-based optimization, mixed integer nonlinear programming, mixed aleatory-epistemic uncertainty quantification, or optimization under uncertainty. By employing object-oriented design to implement abstractions of the key components required for iterative systems analyses, the DAKOTA toolkit provides a flexible problem-solving environment as well as a platform for rapid prototyping of new solution approaches.

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

1.2 Overview of DAKOTA

In the DAKOTA system, the strategy creates and manages iterators and models. In the simplest case, the strategy creates a single iterator and a single model and executes the iterator on the model to perform a single study. In a more advanced case, a hybrid optimization strategy might manage a global optimizer operating on a low-fidelity model in coordination with a local optimizer operating on a high-fidelity model. And on the high end, a surrogate-based optimization under uncertainty strategy would employ an uncertainty quantification iterator nested within an optimization iterator and would employ truth models layered within surrogate models. Thus, iterators and models provide both stand-alone capabilities as well as building blocks for more sophisticated studies.
A model contains a set of variables, an interface, and a set of responses, and the iterator operates on the model to map the variables into responses using the interface. Each of these components is a flexible abstraction with a variety of specializations for supporting different types of iterative studies. In a DAKOTA input file, the user specifies these components through strategy, method, model, variables, interface, and responses keyword specifications.

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

The software components are presented in the following sections using a top-down order.

### 1.2.1 Strategies

Class hierarchy: Strategy.

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

- **SingleMethodStrategy**: the simplest strategy. A single iterator is run on a single model to perform a single study.
- **HybridStrategy**: hybrid minimization using a set of iterators employing a corresponding set of models of varying fidelity. Coordination approaches among the iterators include collaborative, embedded, and sequential approaches, as embodied in the CollaborativeHybridStrategy, EmbeddedHybridStrategy, and SequentialHybridStrategy derived classes.
- **ConcurrentStrategy**: two similar algorithms are available: (1) multi-start iteration from several different starting points, and (2) pareto set optimization for several different multiobjective weightings. Employs a single iterator with a single model, but runs multiple instances of the iterator concurrently for different settings within the model.

### 1.2.2 Iterators

Class hierarchy: Iterator.

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

- **Optimization**: Optimizer provides a base class for the DOTOptimizer, CONMINOptimizer, NPSOLOptimizer, NLPQLPOptimizer, and SNLLOptimizer gradient-based optimization libraries and the APPSOptimizer, COLINOptimizer, JEGAOptimizer, and NCSUOptimizer nongradient-based optimization methods and libraries.
- **Parameter estimation**: LeastSq provides a base class for NL2SOLLeastSq, a least-squares solver based on NL2SOL, SNLLLeastSq, a Gauss-Newton least-squares solver, and NLSSOLLeastSq, an SQP-based least-squares solver.
- **Surrogate-based minimization (optimization and nonlinear least squares)**: SurrBasedMinimizer provides a base class for SurrBasedLocalMinimizer, SurrBasedGlobalMinimizer, and EffGlobalMinimizer. The
surrogate-based local and global methods employ a single iterator with any of the available SurrogateModel capabilities (local, multipoint, or global data fits or hierarchical approximations) and perform a sequence of approximate optimizations, each involving build, optimize, and verify steps. The efficient global method, on the other hand, hard-wires a recursion involving Gaussian process surrogate models coupled with the DIRECT global optimizer to maximize an expected improvement function.

and the Analyzer classes include:

- Uncertainty quantification: NonD provides a base class for non-deterministic methods NonDSampling, NonDReliability (reliability analysis), NonDExpansion (stochastic expansion methods), NonDIntegration (numerical integration methods), and NonDInterval (interval-based epistemic methods). Bayesian calibration methods are prototyped in NonDBayesCal.

  - NonDSampling is further specialized with the NonDLHSSampling class for Latin hypercube and Monte Carlo sampling, the NonDIncLHSSampling class for incremental Latin hypercube sampling, and NonDAImpSampling for multimodal adaptive importance sampling.

  - NonDReliability is further specialized with local and global methods (NonDLocalReliability and NonDGlobalReliability).

  - NonDExpansion includes specializations for generalized polynomial chaos (NonDPolynomialChaos) and stochastic collocation (NonDStochCollocation) and is supported by NonDIntegration, which supplies tensor-product quadrature and Smolyak sparse grid methods (NonDQuadrature and NonDSparseGrid).

  - NonDInterval provides a base class for epistemic interval-based UQ methods. Three interval analysis approaches are provided: LHS (NonDLHSInterval), surrogate-based global optimization (NonDGlobalInterval), and local optimization (NonDLocalInterval). Each of these three has specializations for single interval (NonDLHSSingleInterval, NonDGlobalSingleInterval, NonDLocalSingleInterval) and Dempster-Shafer Theory of Evidence (NonDLHSEvidence, NonDGlobalEvidence, NonDLocalEvidence) approaches.

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

1.2.3 Models

Class hierarchy: Model.

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

- SingleModel: variables are mapped into responses using a single Interface object. No sub-iterators or sub-models are used.
• **SurrogateModel**: variables are mapped into responses using an approximation. The approximation is built and/or corrected using data from a sub-model (the truth model) and the data may be obtained using a sub-iterator (a design of experiments iterator). **SurrogateModel** has two derived classes: **DataFitSurrModel** for data fit surrogates and **HierarchSurrModel** for hierarchical models of varying fidelity. The relationship of the sub-iterations 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-iterations and sub-models is considered to be "nested" since they are used to perform a complete iterative study as part of every response evaluation on the top level model.

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

### 1.2.4 Variables

Class hierarchy: **Variables**.

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

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

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

Whereas domain types are controlled through the derived class selection, variable types are handled within each of these derived classes using variable views. These views control the subset of variable types that are active and inactive within a particular iterative study. For design optimization and uncertainty quantification, for example, the active variables view consists of design or uncertain types, respectively, and any other variable types are carried along invisible to the iterative algorithm being employed. For parameter studies and design of experiments, however, a variable subset view is not imposed and all variables are active. Selected uncertainty quantification methods can also be toggled into an "All" view using the **all_variables** input specification.

Any inactive view is set based on higher level iteration within a model recursion (e.g., a **NestedModel**), which enables lower level iteration to return derivatives with respect to variables that are active at the higher level.

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

Class hierarchy: Interface.

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

- **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

- **DirectApplicInterface**: the simulation is linked into the DAKOTA executable and is invoked using a procedure call. Asynchronous invocations will utilize nonblocking threads (capability not yet available).

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

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

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

1.2.6 Responses

Class: Response.

The Response class provides an abstract data representation of response functions and their first and second derivatives (gradient vectors and Hessian matrices). These response functions can be interpreted as 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 man-
  ages concurrent analyses executing on multiple processors. Partitioning of these levels with MPI communic-
  ators is managed in ParallelLibrary and scheduling routines for the levels are part of Strategy, Application-
  Interface, and ForkApplicInterface.

- Parsing: DAKOTA employs the NIDR parser (New Input Deck Reader) to retrieve information from user
  input files. Parsing options are processed in CommandLineHandler and parsing occurs in ProblemDesc-
  DB::manage_inputs() called from main.C. NIDR uses the keyword handlers in the NIDRProblemDesc-
  DB derived class to populate data within the ProblemDescDB base class, which maintains a DataStrategy
  specification and lists of DataMethod, DataModel, DataVariables, DataInterface, and DataResponses
  specifications. Procedures for modifying the parsing subsystem are described in Instructions for Modifying
  DAKOTA's Input Specification.

- Failure capturing: Simulation failures can be trapped and managed using exception handling in Application-
  Interface 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 interrogat-
ing, converting, repairing, concatenating, and post-processing restart files.

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

- Graphics: DAKOTA provides 2D iteration history graphics using Motif widgets and 3D surface plotting
  graphics from the PLPLOT package. Graphics data can also be catalogued in a tabular data file for post-
  processing with 3rd party tools such as Matlab, Tecplot, etc. All of these capabilities are encapsulated
  within the Graphics class.
1.4 Additional Resources

Additional development resources include:

- Recommended Practices for DAKOTA Development
- Software Tools for DAKOTA Development
- Instructions for Modifying DAKOTA’s Input Specification
- Interfacing with DAKOTA as a Library

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

Chapter 2

DAKOTA Namespace Index

2.1 DAKOTA Namespace List

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

- Dakota (The primary namespace for DAKOTA) .................................................. 29
- SIM (Plug facilities into DAKOTA) ................................................................. 154
Chapter 3

DAKOTA Hierarchical Index

3.1 DAKOTA Class Hierarchy

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

- ActiveSet .................................................. 155
- AnalysisCode ........................................ 158
- ForkAnalysisCode .................................... 329
- SysCallAnalysisCode ................................ 751
- Approximation ........................................... 178
- BasisPolyApproximation ............................... 200
- InterpPolyApproximation .............................. 375
- OrthogPolyApproximation .............................. 613
- GaussProcApproximation .............................. 339
- SurfpackApproximation ................................. 724
- TANA3Approximation .................................. 756
- TaylorApproximation .................................. 758
- APPSEvalMgr ............................................... 189
- Array ....................................................... 195
- BaseConstructor ......................................... 199
- BasisPolynomial .......................................... 205
- LagrangeInterpPolynomial ............................ 414
- OrthogonalPolynomial ................................. 611
  - GenLaguerreOrthogPolynomial ........................ 345
  - HermiteOrthogPolynomial ............................. 358
  - JacobiOrthogPolynomial ............................... 394
  - LaguerreOrthogPolynomial ............................ 416
  - LegendreOrthogPolynomial ............................ 422
  - NumericGenOrthogPolynomial ........................ 601
- BiStream .................................................. 212
- BoStream ................................................. 215
- COLINApplication ....................................... 218
- ColinPoint ................................................. 225
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommandShell</td>
<td>230</td>
</tr>
<tr>
<td>Constraints</td>
<td>243</td>
</tr>
<tr>
<td>MergedConstraints</td>
<td>428</td>
</tr>
<tr>
<td>MixedConstraints</td>
<td>441</td>
</tr>
<tr>
<td>CtelRegexp</td>
<td>255</td>
</tr>
<tr>
<td>DataInterface</td>
<td>267</td>
</tr>
<tr>
<td>DataMethod</td>
<td>268</td>
</tr>
<tr>
<td>DataMethodRep</td>
<td>269</td>
</tr>
<tr>
<td>DataModel</td>
<td>281</td>
</tr>
<tr>
<td>DataModelRep</td>
<td>282</td>
</tr>
<tr>
<td>DataResponses</td>
<td>286</td>
</tr>
<tr>
<td>DataResponsesRep</td>
<td>287</td>
</tr>
<tr>
<td>DataStrategy</td>
<td>291</td>
</tr>
<tr>
<td>DataStrategyRep</td>
<td>292</td>
</tr>
<tr>
<td>DataVariables</td>
<td>295</td>
</tr>
<tr>
<td>DataVariablesRep</td>
<td>297</td>
</tr>
<tr>
<td>ErrorTable</td>
<td>328</td>
</tr>
<tr>
<td>FunctionCompare</td>
<td>338</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>347</td>
</tr>
<tr>
<td>CommandLineHandler</td>
<td>228</td>
</tr>
<tr>
<td>Graphics</td>
<td>351</td>
</tr>
<tr>
<td>Interface</td>
<td>366</td>
</tr>
<tr>
<td>ApplicationInterface</td>
<td>167</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
<td>311</td>
</tr>
<tr>
<td>ParallelDirectApplicInterface</td>
<td>623</td>
</tr>
<tr>
<td>SerialDirectApplicInterface</td>
<td>692</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>331</td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>355</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
<td>753</td>
</tr>
<tr>
<td>ApproximationInterface</td>
<td>185</td>
</tr>
<tr>
<td>Iterator</td>
<td>382</td>
</tr>
<tr>
<td>Analyzer</td>
<td>163</td>
</tr>
<tr>
<td>NonD</td>
<td>518</td>
</tr>
<tr>
<td>NonDBayesCal</td>
<td>530</td>
</tr>
<tr>
<td>NonDExpansion</td>
<td>532</td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>576</td>
</tr>
<tr>
<td>NonDStochCollocation</td>
<td>596</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>548</td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>578</td>
</tr>
<tr>
<td>NonDSparseGrid</td>
<td>590</td>
</tr>
<tr>
<td>NonDInterval</td>
<td>551</td>
</tr>
<tr>
<td>NonDGlobalInterval</td>
<td>537</td>
</tr>
<tr>
<td>NonDGlobalEvidence</td>
<td>535</td>
</tr>
<tr>
<td>NonDGlobalSingleInterval</td>
<td>544</td>
</tr>
<tr>
<td>NonDLHSInterval</td>
<td>556</td>
</tr>
<tr>
<td>NonDLHSEvidence</td>
<td>554</td>
</tr>
<tr>
<td>NonDLHSSingleInterval</td>
<td>561</td>
</tr>
</tbody>
</table>
NonDLocalInterval .................................................. 565
NonDLocalEvidence .................................................. 563
NonDLocalSingleInterval ........................................... 574
NonDReliability ..................................................... 581
NonDGlobalReliability .............................................. 541
NonDLocalReliability .............................................. 568
NonDSampling .......................................................... 585
NonDAadaptImpSampling ........................................... 526
NonDIncremLHSSampling .......................................... 546
NonDLHSSampling ................................................... 558
PStudyDACE ............................................................ 661
DDACEDesignCompExp ............................................. 307
FSUDESIGNCompExp ............................................... 334
ParamStudy ............................................................. 644
PSUADEDesignCompExp ........................................... 664
Minimizer ............................................................... 433
LeastSq ............................................................... 418
NL2SOLLeastSq ..................................................... 507
NLSSOLLeastSq ..................................................... 515
SNLLLeastSq .......................................................... 701
Optimizer ............................................................... 606
APPSOptimizer ....................................................... 192
COLINOptimizer ..................................................... 221
CONMINOptimizer ................................................... 235
DOTOptimizer .......................................................... 318
JEGAOptimizer ....................................................... 396
NCUSOptimizer ....................................................... 491
NLQPLPOptimizer ................................................... 510
NPSOLOptimizer ..................................................... 597
SNLLOptimizer .......................................................... 705
SurrBasedMinimizer .................................................. 736
EffGlobalMinimizer .................................................. 323
SurrBasedGlobalMinimizer ........................................... 727
SurrBasedLocalMinimizer ........................................... 729
JEGAOptimizer::Driver ............................................. 404
JEGAOptimizer::Evaluator ......................................... 406
JEGAOptimizer::EvaluatorCreator ................................. 412
List ............................................................... 424
Model ............................................................... 445
NestedModel .......................................................... 494
RecastModel ............................................................ 668
SingleModel ............................................................ 695
SurrogateModel .......................................................... 745
DataFitSurrModel ................................................... 258
HierarchSurrModel ................................................... 360
Model::FDhelp .......................................................... 484
MPIPackBuffer .......................................................... 485
MPIUnpackBuffer .................................................... 488
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL2Res</td>
<td>506</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>517</td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>621</td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>624</td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>627</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>640</td>
</tr>
<tr>
<td><code>partial_prp_equality</code></td>
<td>649</td>
</tr>
<tr>
<td><code>partial_prp_hash</code></td>
<td>650</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>651</td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
<td>501</td>
</tr>
<tr>
<td>RecastBaseConstructor</td>
<td>667</td>
</tr>
<tr>
<td>Response</td>
<td>674</td>
</tr>
<tr>
<td>SensAnalysisGlobal</td>
<td>679</td>
</tr>
<tr>
<td>SNLLBase</td>
<td>685</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>701</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>705</td>
</tr>
<tr>
<td>SOLBase</td>
<td>711</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>515</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>597</td>
</tr>
<tr>
<td>Strategy</td>
<td>714</td>
</tr>
<tr>
<td>ConcurrentStrategy</td>
<td>232</td>
</tr>
<tr>
<td>HybridStrategy</td>
<td>364</td>
</tr>
<tr>
<td>CollaborativeHybridStrategy</td>
<td>226</td>
</tr>
<tr>
<td>EmbeddedHybridStrategy</td>
<td>326</td>
</tr>
<tr>
<td>SequentialHybridStrategy</td>
<td>687</td>
</tr>
<tr>
<td>SingleMethodStrategy</td>
<td>693</td>
</tr>
<tr>
<td>String</td>
<td>721</td>
</tr>
<tr>
<td>SurrogateDataPoint</td>
<td>741</td>
</tr>
<tr>
<td>SurrogateDataPointRep</td>
<td>743</td>
</tr>
<tr>
<td>TrackerHTTP</td>
<td>760</td>
</tr>
<tr>
<td>Variables</td>
<td>763</td>
</tr>
<tr>
<td>MergedVariables</td>
<td>430</td>
</tr>
<tr>
<td>MixedVariables</td>
<td>443</td>
</tr>
</tbody>
</table>
Chapter 4

DAKOTA Class Index

4.1 DAKOTA Class List

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

- ActiveSet (Active set request vector and the derivative variables vector)..........................155
- AnalysisCode (Processes for managing simulations).........................................................158
- Analyzer (Hierarchy)........................................................................................................163
- ApplicationInterface (Interfaces to simulation codes)....................................................167
- Approximation (Base class for the approximation class hierarchy).................................178
- ApproximationInterface (Approximations to simulation-based results).............................185
- APPSEvalMgr (Evaluation manager class for APPSPACK)................................................189
- APPSOptimizer (Wrapper class for APPSPACK)..............................................................192
- Array (Template class for the Dakota bookkeeping array).................................................195
- BaseConstructor (Dummy struct for overloading letter-envelope constructors)..................199
- BasisPolyApproximation (Derived approximation class for global basis polynomials).......200
- BasisPolynomial (Base class for the basis polynomial class hierarchy)............................205
- BiStream (Data types).......................................................................................................212
- BoStream (Data types).......................................................................................................215
- COLINApplication............................................................................................................218
- COLINOptimizer (Wrapper class for optimizers defined using COLIN)..............................221
- ColinPoint..........................................................................................................................225
- CollaborativeHybridStrategy (Optimization and nonlinear least squares methods)...........226
- CommandLineHandler (Utility class for managing command line inputs to DAKOTA)...........228
- CommandShell (Processes with system calls)......................................................................230
- ConcurrentStrategy (Strategy for multi-start iteration or pareto set optimization)..........232
- CONMINOptimizer (Wrapper class for the CONMIN optimization library).........................235
- Constraints (Base class for the variable constraints class hierarchy)...............................243
- CtelRegexp.......................................................................................................................255
- DataFitSurrModel (Data fit surrogates (global and local))...............................................258
- DataInterface (Handle class for interface specification data)............................................267
- DataMethod (Handle class for method specification data)................................................268
- DataMethodRep (Body class for method specification data).............................................269
- DataModel (Handle class for model specification data).....................................................281
**DAKOTA Class Index**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataModelRep</td>
<td>(Body class for model specification data)</td>
<td>282</td>
</tr>
<tr>
<td>DataResponses</td>
<td>(Handle class for responses specification data)</td>
<td>286</td>
</tr>
<tr>
<td>DataResponsesRep</td>
<td>(Body class for responses specification data)</td>
<td>287</td>
</tr>
<tr>
<td>DataStrategy</td>
<td>(Handle class for strategy specification data)</td>
<td>291</td>
</tr>
<tr>
<td>DataStrategyRep</td>
<td>(Body class for strategy specification data)</td>
<td>292</td>
</tr>
<tr>
<td>DataVariables</td>
<td>(Handle class for variables specification data)</td>
<td>295</td>
</tr>
<tr>
<td>DataVariablesRep</td>
<td>(Body class for variables specification data)</td>
<td>297</td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>(Wrapper class for the DDACE design of experiments library)</td>
<td>307</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
<td>(And testers using direct procedure calls)</td>
<td>311</td>
</tr>
<tr>
<td>DOTOptimizer</td>
<td>(Wrapper class for the DOT optimization library)</td>
<td>318</td>
</tr>
<tr>
<td>EffGlobalMinimizer</td>
<td>(Implementation of Efficient Global Optimization/Least Squares algorithms)</td>
<td>323</td>
</tr>
<tr>
<td>EmbeddedHybridStrategy</td>
<td>(Search methods)</td>
<td>326</td>
</tr>
<tr>
<td>ErrorTable</td>
<td>(Data structure to hold errors)</td>
<td>328</td>
</tr>
<tr>
<td>ForkAnalysisCode</td>
<td>(Simulations using forks)</td>
<td>329</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>(Using forks)</td>
<td>331</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>(Wrapper class for the FSUDace QMC/CVT library)</td>
<td>334</td>
</tr>
<tr>
<td>FunctionCompare</td>
<td></td>
<td>338</td>
</tr>
<tr>
<td>GaussProcApproximation</td>
<td>(Derived approximation class for Gaussian Process implementation)</td>
<td>339</td>
</tr>
<tr>
<td>GenLaguerreOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for generalized Laguerre polynomials)</td>
<td>345</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>((Advanced Computer Research Institute, Lyon, France))</td>
<td>347</td>
</tr>
<tr>
<td>Graphics</td>
<td>(For post-processing with Matlab, Tecplot, etc)</td>
<td>351</td>
</tr>
<tr>
<td>GridApplicInterface</td>
<td>(Using grid services such as Condor or Globus)</td>
<td>355</td>
</tr>
<tr>
<td>HermiteOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Hermite polynomials)</td>
<td>358</td>
</tr>
<tr>
<td>HierarchSurrModel</td>
<td>(Hierarchical surrogates (models of varying fidelity))</td>
<td>360</td>
</tr>
<tr>
<td>HybridStrategy</td>
<td>(Base class for hybrid minimization strategies)</td>
<td>364</td>
</tr>
<tr>
<td>Interface</td>
<td>(Base class for the interface class hierarchy)</td>
<td>366</td>
</tr>
<tr>
<td>InterpPolyApproximation</td>
<td>(Approximation)</td>
<td>375</td>
</tr>
<tr>
<td>Iterator</td>
<td>(Base class for the iterator class hierarchy)</td>
<td>382</td>
</tr>
<tr>
<td>JacobiOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Jacobi polynomials)</td>
<td>394</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>(A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA))</td>
<td>396</td>
</tr>
<tr>
<td>JEGAOptimizer::Driver</td>
<td>(A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm)</td>
<td>404</td>
</tr>
<tr>
<td>JEGAOptimizer::Evaluator</td>
<td>(An evaluator specialization that knows how to interact with Dakota)</td>
<td>406</td>
</tr>
<tr>
<td>JEGAOptimizer::EvaluatorCreator</td>
<td>(A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of an Evaluator)</td>
<td>412</td>
</tr>
<tr>
<td>LagrangeInterpPolynomial</td>
<td>(Derived basis polynomial class for 1-D Lagrange interpolation polynomials)</td>
<td>414</td>
</tr>
<tr>
<td>LaguerreOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Laguerre polynomials)</td>
<td>416</td>
</tr>
<tr>
<td>LeastSq</td>
<td>(Base class for the nonlinear least squares branch of the iterator hierarchy)</td>
<td>418</td>
</tr>
<tr>
<td>LegendreOrthogPolynomial</td>
<td>(Derived orthogonal polynomial class for Legendre polynomials)</td>
<td>422</td>
</tr>
<tr>
<td>List</td>
<td>(Template class for the Dakota bookkeeping list)</td>
<td>424</td>
</tr>
<tr>
<td>MergedConstraints</td>
<td>(Merged data view)</td>
<td>428</td>
</tr>
<tr>
<td>MergedVariables</td>
<td>(Merged data view)</td>
<td>430</td>
</tr>
<tr>
<td>Minimizer</td>
<td>(Iterator hierarchy)</td>
<td>433</td>
</tr>
<tr>
<td>MixedConstraints</td>
<td>(Default data view (no variable or domain type array merging))</td>
<td>441</td>
</tr>
<tr>
<td>MixedVariables</td>
<td>(Default data view (no variable or domain type array merging))</td>
<td>443</td>
</tr>
<tr>
<td>Model</td>
<td>(Base class for the model class hierarchy)</td>
<td>445</td>
</tr>
<tr>
<td>Model::FDhelp</td>
<td>(Possibly adjusted for bounds)</td>
<td>484</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>Class for packing MPI message buffers</td>
<td>485</td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>Class for unpacking MPI message buffers</td>
<td>488</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>Wrapper class for the NCSU DIRECT optimization library</td>
<td>491</td>
</tr>
<tr>
<td>NestedModel</td>
<td>(Execution within every evaluation of the model)</td>
<td>494</td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
<td>(The derived input file database utilizing the new IDR parser)</td>
<td>501</td>
</tr>
<tr>
<td>NL2Res</td>
<td>(Auxiliary information passed to calcrl and calcU via ur)</td>
<td>506</td>
</tr>
<tr>
<td>NL2SOLLeastSq</td>
<td>(Wrapper class for the NL2SOL nonlinear least squares library)</td>
<td>507</td>
</tr>
<tr>
<td>NLPQLPOptimizer</td>
<td>(Wrapper class for the NLPQLP optimization library, Version 2.0)</td>
<td>510</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>(Wrapper class for the NLSSOL nonlinear least squares library)</td>
<td>515</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>(Dummy struct for overloading constructors used in on-the-fly instantiations)</td>
<td>517</td>
</tr>
<tr>
<td>NonD</td>
<td>(Base class for all nondeterministic iterators (the DAKOTA/UQ branch))</td>
<td>518</td>
</tr>
<tr>
<td>NonDAdaptImpSampling</td>
<td>(Class for the Adaptive Importance Sampling methods within DAKOTA)</td>
<td>526</td>
</tr>
<tr>
<td>NonDBayesCal</td>
<td>(Generates posterior distribution on model parameters given experiment data)</td>
<td>530</td>
</tr>
<tr>
<td>NonDExpansion</td>
<td>(Colllocation (SC))</td>
<td>532</td>
</tr>
<tr>
<td>NonDGlobalEvidence</td>
<td>(Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ)</td>
<td>535</td>
</tr>
<tr>
<td>NonDGlobalInterval</td>
<td>(To calculate interval bounds for epistemic uncertainty quantification)</td>
<td>537</td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
<td>(Class for global reliability methods within DAKOTA/UQ)</td>
<td>541</td>
</tr>
<tr>
<td>NonDGlobalSingleInterval</td>
<td>(To calculate interval bounds for epistemic uncertainty quantification)</td>
<td>544</td>
</tr>
<tr>
<td>NonDIncrementLHSSampling</td>
<td>(Performs incremental LHS sampling for uncertainty quantification)</td>
<td>546</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>(Numerical integration points for evaluation of expectation integrals)</td>
<td>548</td>
</tr>
<tr>
<td>NonDInterval</td>
<td>(Base class for interval-based methods within DAKOTA/UQ)</td>
<td>551</td>
</tr>
<tr>
<td>NonDLHSEvidence</td>
<td>(Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ)</td>
<td>554</td>
</tr>
<tr>
<td>NonDLHSInterval</td>
<td>(Class for the LHS-based interval methods within DAKOTA/UQ)</td>
<td>556</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>(Performs LHS and Monte Carlo sampling for uncertainty quantification)</td>
<td>558</td>
</tr>
<tr>
<td>NonDLHSSingleInterval</td>
<td>(Class for pure interval propagation using LHS)</td>
<td>561</td>
</tr>
<tr>
<td>NonDLLocalEvidence</td>
<td>(Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ)</td>
<td>563</td>
</tr>
<tr>
<td>NonDLLocalInterval</td>
<td>(Calculate interval bounds for epistemic uncertainty quantification)</td>
<td>565</td>
</tr>
<tr>
<td>NonDLLocalReliability</td>
<td>(Class for the reliability methods within DAKOTA/UQ)</td>
<td>568</td>
</tr>
<tr>
<td>NonDLLocalSingleInterval</td>
<td>(Calculate interval bounds for epistemic uncertainty quantification)</td>
<td>574</td>
</tr>
<tr>
<td>NonDPolynomialChaos</td>
<td>(Quantification)</td>
<td>576</td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>(Normals/uniforms/exponentials/betas/gammas)</td>
<td>578</td>
</tr>
<tr>
<td>NonDReliability</td>
<td>(Base class for the reliability methods within DAKOTA/UQ)</td>
<td>581</td>
</tr>
<tr>
<td>NonDSampling</td>
<td>(NonDIncrementLHSSampling, and NonDAdaptImpSampling)</td>
<td>585</td>
</tr>
<tr>
<td>NonDSparseGrid</td>
<td>(Integrals over independent standard random variables)</td>
<td>590</td>
</tr>
<tr>
<td>NonDStochasticCollocation</td>
<td>(Quantification)</td>
<td>596</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>(Wrapper class for the NPSOL optimization library)</td>
<td>597</td>
</tr>
<tr>
<td>NumericGenOrthogPolynomial</td>
<td>(Orthogonal polynomials)</td>
<td>601</td>
</tr>
<tr>
<td>Optimizer</td>
<td>(Base class for the optimizer branch of the iterator hierarchy)</td>
<td>606</td>
</tr>
<tr>
<td>OrthogonalPolynomial</td>
<td>(Base class for the orthogonal polynomial class hierarchy)</td>
<td>611</td>
</tr>
<tr>
<td>OrthogPolyApproximation</td>
<td>(Approximation)</td>
<td>613</td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>(Collectively identify a particular multilevel parallel configuration)</td>
<td>621</td>
</tr>
<tr>
<td>ParallelDirectApplicInterface</td>
<td>(Plug-ins using assign_rep())</td>
<td>623</td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>(Communicator partitioning)</td>
<td>624</td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>(Message passing within these levels)</td>
<td>627</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>(Evaluation id)</td>
<td>640</td>
</tr>
<tr>
<td>ParamStudy</td>
<td>(Class for vector, list, centered, and multidimensional parameter studies)</td>
<td>644</td>
</tr>
<tr>
<td>partial_prp_equality</td>
<td>(Predicate for comparing ONLY the idInterface and Vars attributes of PRPair)</td>
<td>649</td>
</tr>
<tr>
<td>partial_prp_hash</td>
<td>(Wrapper to delegate to the ParamResponsePair hash_value function)</td>
<td>650</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>(The database containing information parsed from the DAKOTA input file)</td>
<td>651</td>
</tr>
</tbody>
</table>
DAKOTA Class Index

PStudyDACE (Design of experiments methods) ........................................ 661
PSUADEDesignCompExp (Wrapper class for the PSUADE library) .................. 664
RecastBaseConstructor (Instantiations) .................................................. 667
RecastModel (In order to recast the form of its inputs and/or outputs) ........... 668
Response (Response provides the handle class) ....................................... 674
ResponseRep (ResponseRep provides the body class) ................................ 679
SensAnalysisGlobal (And variance-based decomposition) .......................... 685
SequentialHybridStrategy (Models of varying fidelity) .............................. 687
SerialDirectApplicInterface (Plug-ins using assign_rep()) .......................... 692
SingleMethodStrategy (Single model) .................................................... 693
SingleModel (Variables into responses) .................................................. 695
SNLLBase (Base class for OPT++ optimization and least squares methods) ...... 698
SNLLLeastSq (Wrapper class for the OPT++ optimization library) ............... 701
SNLLOptimizer (Wrapper class for the OPT++ optimization library) .......... 705
SOLBase (Base class for Stanford SOL software) ...................................... 711
Strategy (Base class for the strategy class hierarchy) .............................. 714
String (Dakota::String class, used as main string class for Dakota) .......... 721
SurfpackApproximation (Interface between Surfpack and Dakota) .............. 724
SurrBasedGlobalMinimizer (And updates a global surrogate model without trust region controls) .................................................. 727
SurrBasedLocalMinimizer (And nonlinear least squares) ............................ 729
SurrBasedMinimizer (Base class for local/global surrogate-based optimization/least squares) .................................................. 736
SurrogateDataPoint (For defining a "truth" data point) ............................. 741
SurrogateDataPointRep (Or body, may be shared by multiple SurrogateDataPoint handle instances) .................................................. 743
SurrogateModel (Base class for surrogate models (DataFitSurrModel and HierarchSurrModel)) .................................................. 745
SysCallAnalysisCode (Simulations using system calls) .............................. 751
SysCallApplicInterface (Using system calls) .......................................... 753
TANA3Approximation (Approximation (a multipoint approximation)) .......... 756
TaylorApproximation (Series (a local approximation)) ............................. 758
TrackerHTTP (Curl library) .................................................................... 760
Variables (Base class for the variables class hierarchy) ............................ 763
Chapter 5

DAKOTA File Index

5.1 DAKOTA File List

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

- `dll_api.C` (This file contains a DakotaRunner class, which launches DAKOTA) ............... 777
- `dll_api.h` (API for DLL interactions) ................................................................. 779
- `JEGAOptimizer.C` (Contains the implementation of the JEGAOptimizer class) ............. 781
- `JEGAOptimizer.H` (Contains the definition of the JEGAOptimizer class) .................. 782
- `library_mode.C` (File containing a mock simulator main for testing DAKOTA in library mode) ................. 783
- `main.C` (File containing the main program for DAKOTA) ...................................... 785
- `restart_util.C` (File containing the DAKOTA restart utility main program) .................... 786
Chapter 6

DAKOTA Page Index

6.1 DAKOTA Related Pages

Here is a list of all related documentation pages:

- Recommended Practices for DAKOTA Development ............................................. 787
- Instructions for Modifying DAKOTA’s Input Specification ...................................... 791
- Interfacing with DAKOTA as a Library ................................................................. 797
- Performing Function Evaluations ........................................................................ 807
- Software Tools for DAKOTA Development ......................................................... 809
- Todo List .............................................................................................................. ??
Chapter 7

DAKOTA Namespace Documentation

7.1 Dakota Namespace Reference

The primary namespace for DAKOTA.

Classes

- class AnalysisCode
  processes for managing simulations.

- class ApplicationInterface
  interfaces to simulation codes.

- class ApproximationInterface
  approximations to simulation-based results.

- class APPSEvalMgr
  Evaluation manager class for APPSPACK.

- class APPSOptimizer
  Wrapper class for APPSPACK.

- class BasisPolyApproximation
  Derived approximation class for global basis polynomials.

- class BasisPolynomial
  Base class for the basis polynomial class hierarchy.

- class COLINApplication
- class COLINOptimizer
  Wrapper class for optimizers defined using COLIN.
• class **CollaborativeHybridStrategy**
  optimization and nonlinear least squares methods.

• class **GetLongOpt**
  (Advanced Computer Research Institute, Lyon, France).

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

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

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

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

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

• class **Analyzer**
  hierarchy.

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

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

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

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

• class **BiStream**
  data types

• class **BoStream**
  data types

• class **Constraints**
  Base class for the variable constraints class hierarchy.
• class Graphics
  for post-processing with Matlab, Tecplot, etc.

• class Interface
  Base class for the interface class hierarchy.

• class Iterator
  Base class for the iterator class hierarchy.

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

• class List
  Template class for the Dakota bookkeeping list.

• class FunctionCompare
• class Minimizer
  iterator hierarchy.

• class Model
  Base class for the model class hierarchy.

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

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

• class PStudyDACE
  design of experiments methods.

• class ResponseRep
  ResponseRep provides the body class.

• class Response
  Response provides the handle class.

• class Strategy
  Base class for the strategy class hierarchy.

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

• class Variables
  Base class for the variables class hierarchy.
• class DataFitSurrModel
  data fit surrogates (global and local)

• class DataInterface
  Handle class for interface specification data.

• class DataMethodRep
  Body class for method specification data.

• class DataMethod
  Handle class for method specification data.

• class DataModelRep
  Body class for model specification data.

• class DataModel
  Handle class for model specification data.

• class DataResponsesRep
  Body class for responses specification data.

• class DataResponses
  Handle class for responses specification data.

• class DataStrategyRep
  Body class for strategy specification data.

• class DataStrategy
  Handle class for strategy specification data.

• class DataVariablesRep
  Body class for variables specification data.

• class DataVariables
  Handle class for variables specification data.

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

• class DirectApplicInterface
  and testers using direct procedure calls.

• class DOTOptimizer
  Wrapper class for the DOT optimization library.

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

- class **EmbeddedHybridStrategy**
  search methods.

- class **ForkAnalysisCode**
  simulations using forks.

- class **ForkApplicInterface**
  using forks.

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

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

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

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

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

- struct **RecastBaseConstructor**
  instantiations.

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

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

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

- class **HybridStrategy**
  Base class for hybrid minimization strategies.

- class **InterpPolyApproximation**
  approximation.

- class **JacobiOrthogPolynomial**
  Derived orthogonal polynomial class for Jacobi polynomials.
• class JEGAOptimizer
  A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

• class LagrangeInterpPolynomial
  Derived basis polynomial class for 1-D Lagrange interpolation polynomials.

• class LaguerreOrthogPolynomial
  Derived orthogonal polynomial class for Laguerre polynomials.

• class LegendreOrthogPolynomial
  Derived orthogonal polynomial class for Legendre polynomials.

• class MergedConstraints
  the merged data view.

• class MergedVariables
  merged data view.

• class MixedConstraints
  the default data view (no variable or domain type array merging).

• class MixedVariables
  the default data view (no variable or domain type array merging).

• class MPIPackBuffer
  Class for packing MPI message buffers.

• class MPIUnpackBuffer
  Class for unpacking MPI message buffers.

• class NCSUOptimizer
  Wrapper class for the NCSU DIRECT optimization library.

• class NestedModel
  execution within every evaluation of the model.

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

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

• class NL2SOLLeastSq
  Wrapper class for the NL2SOL nonlinear least squares library.
• class NLPQLPOptimizer
  Wrapper class for the NLPQLP optimization library, Version 2.0.

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

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

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

• class NonDExpansion
  collocation (SC)

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

• class NonDGlobalInterval
  to calculate interval bounds for epistemic uncertainty quantification

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

• class NonDGlobalSingleInterval
  to calculate interval bounds for epistemic uncertainty quantification

• class NonDIncremLHSSampling
  Performs incremental LHS sampling for uncertainty quantification.

• class NonDIIntegration
  numerical integration points for evaluation of expectation integrals

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

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

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

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

• class NonDLHSSingleInterval
Class for pure interval propagation using LHS.

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

- class **NonDLocalInterval**
  
  calculate interval bounds for epistemic uncertainty quantification

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

- class **NonDLocalSingleInterval**
  
  calculate interval bounds for epistemic uncertainty quantification

- class **NonDPolynomialChaos**
  
  quantification

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

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

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

- class **NonDSparseGrid**
  
  integrals over independent standard random variables.

- class **NonDStochCollocation**
  
  quantification

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

- class **NumericGenOrthogPolynomial**
  
  orthogonal polynomials

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

- class **OrthogPolyApproximation**
  
  approximation).

- class **ParallelLevel**
  
  communicator partitioning.
7.1 Dakota Namespace Reference

- class `ParallelConfiguration`
  collectively identify a particular multilevel parallel configuration.

- class `ParallelLibrary`
  message passing within these levels.

- class `ParamResponsePair`
  evaluation id.

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

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

- struct `partial_prp_hash`
  wrapper to delegate to the `ParamResponsePair hash_value` function.

- struct `partial_prp_equality`
  predicate for comparing ONLY the idInterface and Vars attributes of PRPair.

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

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

- class `SensAnalysisGlobal`
  and variance-based decomposition.

- class `SequentialHybridStrategy`
  models of varying fidelity.

- class `SingleMethodStrategy`
  single model.

- class `SingleModel`
  variables into responses.

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

- class `SNLLLeastSq`
  Wrapper class for the OPT++ optimization library.
• class SNLLOptimizer
  Wrapper class for the OPT++ optimization library.

• class SOLBase
  Base class for Stanford SOL software.

• class SurfpackApproximation
  Interface between Surfpack and Dakota.

• class SurrBasedGlobalMinimizer
  and updates a global surrogate model without trust region controls

• class SurrBasedLocalMinimizer
  and nonlinear least squares.

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

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

• class SysCallAnalysisCode
  simulations using system calls.

• class SysCallApplicInterface
  using system calls.

• class TANA3Approximation
  approximation (a multipoint approximation).

• class TaylorApproximation
  series (a local approximation).

• class TrackerHTTP
  curl library

**Typedefs**

• typedef double Real
• typedef Teuchos::SerialDenseVector< int, Real > RealVector
• typedef Teuchos::SerialDenseVector< int, int > IntVector
• typedef Teuchos::SerialDenseMatrix< int, Real > RealMatrix
• typedef Teuchos::SerialSymDenseMatrix< int, Real > RealSymMatrix
• typedef Teuchos::SerialDenseSolver< int, Real > RealSolver
• typedef Teuchos::SerialSpdDenseSolver< int, Real > RealSpdSolver
• typedef std::deque<bool> BoolDeque
• typedef Array<BoolDeque> BoolDequeArray
• typedef Array<Real> RealArray
• typedef Array<RealArray> Real2DArray
• typedef Array<int> IntArray
• typedef Array<IntArray> Int2DArray
• typedef Array<unsigned int> UIntArray
• typedef Array<short> ShortArray
• typedef Array<unsigned short> UShortArray
• typedef Array<UShortArray> UShort2DArray
• typedef Array<UShort2DArray> UShort3DArray
• typedef Array<size_t> SizetArray
• typedef Array<SizetArray> Sizet2DArray
• typedef Array<String> StringArray
• typedef boost::multi_array_types::index_range idx_range
• typedef boost::multi_array<String, 1> StringMultiArray
• typedef StringMultiArray::array_view<1>::type StringMultiArrayView
• typedef StringMultiArray::const_array_view<1>::type StringMultiArrayConstView
• typedef boost::multi_array<unsigned int, 1> UIntMultiArray
• typedef UIntMultiArray::array_view<1>::type UIntMultiArrayView
• typedef UIntMultiArray::const_array_view<1>::type UIntMultiArrayConstView
• typedef Array<RealVector> RealVectorArray
• typedef Array<RealVectorArray> RealVector2DArray
• typedef Array<RealMatrix> RealMatrixArray
• typedef Array<RealSymMatrix> RealSymMatrixArray
• typedef Array<intVector> IntVectorArray
• typedef Array<Variables> VariablesArray
• typedef Array<Response> ResponseArray
• typedef Array<ParamResponsePair> PRPArray
• typedef Array<Model> ModelArray
• typedef Array<Iterator> IteratorArray
• 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<Interface> InterfaceList
• typedef List<Response> ResponseList
• typedef List<Model> ModelList
• typedef List<Iterator> IteratorList
• typedef std::pair<int, String> IntStringPair
• typedef std::pair<Real, Real> RealRealPair
• typedef std::set<Real> RealSet
• typedef std::set<int> IntSet
• typedef `Array< RealSet >` `RealSetArray`
• typedef `Array< IntSet >` `IntSetArray`
• typedef `std::map< int, int >` `IntIntMap`
• typedef `std::map< int, RealVector >` `IntRealVectorMap`
• typedef `std::map< int, ActiveSet >` `IntActiveSetMap`
• typedef `std::map< int, Variables >` `IntVariablesMap`
• typedef `std::map< int, Response >` `IntResponseMap`
• typedef `std::map< IntArray, size_t >` `IntArraySizeMap`
• typedef `std::map< RealRealPair, ParamResponsePair >` `RealRealParamRespMap`
• typedef `IntList::iterator` `ILIter`
• typedef `IntList::const_iterator` `ILCIter`
• typedef `SizetList::iterator` `StLIter`
• typedef `SizetList::const_iterator` `StLCIter`
• typedef `RealList::iterator` `RLIter`
• typedef `RealList::const_iterator` `RLCIter`
• typedef `StringList::iterator` `StringLIter`
• typedef `StringList::const_iterator` `StringLCIter`
• typedef `VariablesList::iterator` `VarsLIter`
• typedef `InterfaceList::iterator` `InterfLIter`
• typedef `ResponseList::iterator` `RespLIter`
• typedef `ModelList::iterator` `ModelLIter`
• typedef `IteratorList::iterator` `IterLIter`
• typedef `List< ParallelLevel >::iterator` `ParLevLIter`
• typedef `List< ParallelConfiguration >::iterator` `ParConfigLIter`
• typedef `IntSet::iterator` `ISIter`
• typedef `IntSet::const_iterator` `ISCIter`
• typedef `IntIntMap::iterator` `IntIntMIter`
• typedef `IntIntMap::const_iterator` `IntIntMCIter`
• typedef `IntShortMap::iterator` `IntShMIter`
• typedef `IntRealVectorMap::iterator` `IntRDVMIter`
• typedef `IntRealVectorMap::const_iterator` `IntRDVMCIter`
• typedef `IntActiveSetMap::iterator` `IntASMIter`
• typedef `IntVariablesMap::iterator` `IntVarsMIter`
• typedef `IntVariablesMap::const_iterator` `IntVarsMCIter`
• typedef `IntResponseMap::iterator` `IntRespMIter`
• typedef `IntResponseMap::const_iterator` `IntRespMCIter`

- typedef `int (*)( void *) dl_find_optimum_t ( void *, Optimizer1 *, char * )`
  
  definition of start grid computing type (function pointer)

- typedef `int (*)( char *) start_grid_computing_t ( char *analysis_driver_script, char *params_file, char *results_file )`
  
  definition of start grid computing type (function pointer)

- typedef `int (*)( char *) perform_analysis_t ( char *iteration_num )`
  
  definition of perform analysis type (function pointer)

- typedef `int (*)( ) get_jobs_completed_t ()`
7.1 Dakota Namespace Reference

- **typedef int(\*) stop\_grid\_computing\_t ()**
  
  definition of stop grid computing type (function pointer)

- **typedef unsigned char u\_char**
- **typedef unsigned short u\_short**
- **typedef unsigned int u\_int**
- **typedef unsigned long u\_long**
- **typedef long long long\_long**
- **typedef unsigned long UL**
- **typedef void(\*) Calcrj (int *n, int *p, Real *x, Real *r, int *ui, void *ur, Vf vf)**
- **typedef void(\*) Vf ()**
- **typedef Real(\*) NGFPType (const Real &x, const RealVector &params)**
  
  pointer to a PDF evaluation function used within integral evaluators

- **typedef bmi::multi_index_container< Dakota::ParamResponsePair, bmi::indexed_by< bmi::ordered_\-unique< bmi::tag< ordered >, bmi::const_mem_fun< Dakota::ParamResponsePair, const IntStringPair &&Dakota::ParamResponsePair::eval\_interface\_ids > >, bmi::hashed_non_unique< bmi::tag< hashed >, bmi::identity< Dakota::ParamResponsePair >, partial\_prp\_hash, partial\_prp\_equality > > > PRPMultiIndexCache**
  
  Boost Multi-Index Container for caching ParamResponsePairs.

- **typedef PRPMultiIndexCache PRPCache**
- **typedef PRPCache::index\_iterator< ordered >::type PRPCacheOIter**
- **typedef PRPCache::index\_const\_iterator< ordered >::type PRPCacheOCIter**
- **typedef PRPCache::index\_iterator< hashed >::type PRPCacheHIter**
- **typedef PRPCache::index\_const\_iterator< hashed >::type PRPCacheHCIter**
- **typedef PRPCacheOIter PRPCacheIter**
- **typedef PRPCacheOCIter PRPCacheCIter**

  Boost Multi-Index Container for queueing ParamResponsePairs.

- **typedef PRPMultiIndexQueue PRPQueue**
- **typedef PPRQueue::index\_iterator< ordered >::type PRPQueueOIter**
- **typedef PPRQueue::index\_const\_iterator< ordered >::type PRPQueueOCIter**
- **typedef PPRQueue::index\_iterator< hashed >::type PRPQueueHIter**
- **typedef PPRQueue::index\_const\_iterator< hashed >::type PRPQueueHCIter**
- **typedef PPRQueueOIter PPRQueueIter**
- **typedef PPRQueueOCIter PPRQueueCIter**
Enumerations

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

- enum {
  SILENT_OUTPUT, QUIET_OUTPUT, NORMAL_OUTPUT, VERBOSE_OUTPUT,
  DEBUG_OUTPUT }

- enum { STD_NORMAL_U, ASKEY_U, EXTENDED_U }

- enum { NO_REFINE, IS, AIS, MMAIS }

- enum { PROBABILITIES, RELIABILITIES, GEN_RELIABILITIES }

- enum { IGNORE_RANKS, SET_RANKS, GET_RANKS, SET_GET_RANKS }

- enum {
  UNCERTAIN, UNCERTAIN_UNIFORM, ACTIVE, ACTIVE_UNIFORM,
  ALL, ALL_UNIFORM }

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

- enum { BREITUNG, HOHENRACK, HONG }

- enum { EGRA_X, EGRA_U }

- enum { ORIGINAL_PRIMARY, SINGLE_OBJECTIVE, LAGRANGIAN_OBJECTIVE,
  AUGMENTED_LAGRANGIAN_OBJECTIVE }

- enum { NO_CONSTRAINTS, LINEARIZED_CONSTRAINTS, ORIGINAL_CONSTRAINTS }

- enum { NO_RELAX, HOMOTOPY, COMPOSITE_STEP }

- enum { PENALTY_MERIT, ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT,
  AUGMENTED_LAGRANGIAN_MERIT }

- enum { FILTER, TR_RATIO }

- enum { SCALE_NONE, SCALE_VALUE, SCALE_LOG }

- enum { CDV, LINEAR, NONLIN, FN_LSQ }

- enum { DISALLOW, TARGET, BOUNDS }

- enum {
  HERMITE, LEGENDRE, LAGUERRE, JACOBI,
  GENERALIZED_LAGUERRE, NUMERICALLY_GENERATED, LAGRANGE }

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

- enum { QUADRATURE, SPARSE_GRID, REGRESSION, SAMPLING }

  solution approaches for calculating the polynomial chaos coefficients

- enum {
  EMPTY, MERGED_ALL, MIXED_ALL, MERGED_DISTINCT_DESIGN,
  MERGED_DISTINCT_UNCERTAIN, MERGED_DISTINCT_ALEATORY_UNCERTAIN,
  MERGED_DISTINCT_EPISTEMIC_UNCERTAIN, MERGED_DISTINCT_STATE,
  MIXED_DISTINCT_DESIGN, MIXED_DISTINCT_UNCERTAIN, MIXED_DISTINCT_-
  ALEATORY_UNCERTAIN, MIXED_DISTINCT_EPISTEMIC_UNCERTAIN,
  MIXED_DISTINCT_STATE }
enum var_t {
  VAR_x1, VAR_x2, VAR_x3, VAR_b,
  VAR_h, VAR_P, VAR_M, VAR_Y,
  VAR_w, VAR_t, VAR_R, VAR_E,
  VAR_X, VAR_Fs, VAR_P1, VAR_P2,
  VAR_P3, VAR_B, VAR_D, VAR_H,
  VAR_F0, VAR_d
}

enumeration of possible variable types (to index to names)

enum driver_t {
  NO_DRIVER = 0, CANTILEVER_BEAM, CYLINDER_HEAD, EXTENDED_ROSENBROCK,
  GENERALIZED_ROSENBROCK, ROSENBROCK, LOGNORMAL_RATIO, MULTIMODAL,
  PLUGIN_ROSENBROCK, PLUGIN_TEXT_BOOK, SHORT_COLUMN, SOBOL_RATIONAL,
  SOBOL_G_FUNCTION, SOBOL_ISHIGAMI, STEEL_COLUMN_COST, STEEL_COLUMN_-
  PERFORMANCE, TEXT_BOOK, TEXT_BOOK1, TEXT_BOOK2, TEXT_BOOK3,
  TEXT_BOOK_OUU, SALINAS, MODELCENTER, MATLAB,
  PYTHON
}

enumeration of possible driver types (to index to names)

enum local_data_t { VARIABLES_MAP = 1, VARIABLES_VECTOR = 2 }

a bit representation)

enum { SETUP_MODEL, SETUP_USERFUNC }

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

enum {
  DAUIVar_poisson = 0, DAUIVar_binomial = 1, DAUIVar_negative_binomial = 2, DAUIVar_-n
  geometric = 3,
  DAUIVar_hypergeometric = 4, DAUIVar_Nkinds = 5 }

enum { DAUVar_histogram_point = 0, DAURVar_Nkinds = 1 }

enum { CEUVar_interval = 0, CEUVar_Nkinds = 1 }

enum {
  DiscSetVar_design_set_int = 0, DiscSetVar_design_set_real = 1, DiscSetVar_state_set_int = 2, Disc-
  SetVar_state_set_real = 3,
  DiscSetVar_Nkinds = 4 }

enum { N_VLS = 4 }
• enum CG_UPDATETYPE {
  CG_STEEPEST, CG_FLETCHER_REEVES, CG_POLAK_RIBIERE, CG_POLAK_RIBIERE_-
  PLUS,
  CG_HESTENES_STIEFEL }
NonlinearCG update options.

• enum CG_LINESEARCHTYPE { CG_FIXED_STEP, CG_LS_SIMPLE, CG_LS_BRENT, CG_LS_-
  WOLFE }
NonlinearCG linesearch options.

• enum { TOTAL_ORDER, TENSOR_PRODUCT, HEURISTIC_TOTAL_ORDER, TENSOR_-
  PRODUCT_SUM }

• enum {
  LIST = 1, VECTOR_SV, VECTOR_FP, CENTERED,
  MULTIDIM }

• enum EvalType { NLFEvaluator, CONEvaluator }
  enumeration for the type of evaluator function

• enum {
  TH_SILENT_OUTPUT, TH_QUIET_OUTPUT, TH_NORMAL_OUTPUT, TH_VERBOSE_-
  OUTPUT,
  TH_DEBUG_OUTPUT }

Functions

• static const char * basename (const char *s)
  return name of s, stripped of any leading path information

• static void cleanup_and_abort (const String &resname)
  output error message about results file and call abort

• CommandShell & flush (CommandShell &shell)
  convenient shell manipulator function to "flush" the shell

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

• std::istream & operator>>(std::istream &s, ActiveSet &set)
  std::istream extraction operator for ActiveSet. Calls read(std::istream&).

• std::ostream & operator<<(std::ostream &s, const ActiveSet &set)
  std::ostream insertion operator for ActiveSet. Calls write(std::ostream&).

• BiStream & operator>>(BiStream &s, ActiveSet &set)
7.1 Dakota Namespace Reference

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

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

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ActiveSet &set)
  Calls read(MPIUnpackBuffer&).

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const ActiveSet &set)
  MPIPackBuffer insertion operator for ActiveSet. Calls write(MPIPackBuffer&).

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

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

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

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

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

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

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

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

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

- bool interface_id_compare (const Interface &interface, const void *id)
  global comparison function for Interface

- bool method_id_compare (const Iterator &iterator, const void *id)
  global comparison function for Iterator

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

• bool model_id_compare (const Model &model, const void *id)
  global comparison function for Model

• bool operator==( const ResponseRep &rep1, const ResponseRep &rep2)
  equality operator

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

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

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

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

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

• MPIUnpackBuffer & operator>>( MPIUnpackBuffer &s, Response &response)
  read(MPIUnpackBuffer&).

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

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

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

• MPIPackBuffer & operator<<( MPIPackBuffer &s, const String &data)
  Reads String from buffer.

• MPIUnpackBuffer & operator>>( MPIUnpackBuffer &s, String &data)
  Writes String to buffer.
7.1 Dakota Namespace Reference

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

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

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

- **String operator+** (const std::string &s1, const String &s2)
  
  Append a String to a std::string and return the resulting String.

- **String operator+** (const String &s1, const std::string &s2)
  
  Append a std::string to a String and return the resulting String.

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

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

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

- **bool binary_equal_to** (const Variables &vars1, const Variables &vars2)
  
  binary_equal_to (since 'operator==' is not suitable for boost/hash_set)

- **std::size_t hash_value** (const Variables &vars)
  
  hash_value

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

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

- **std::ostream & operator<<** (std::ostream &s, const Variables &vars)
  
  std::ostream insertion operator for Variables.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- template<typename OrdinalType, typename ScalarType> void read_data_annotated (std::istream &s, Teuchos::SerialDenseVector< OrdinalType, ScalarType > &v, StringMultiArray &label_array)
  annotated istream extraction operator for full SerialDenseVector with labels

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

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

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

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

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

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

* aprepro ostream insertion operator for partial SerialDenseVector with labels

• template<typename OrdinalType, typename ScalarType> void write_data_annotated (std::ostream &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, const StringMultiArray &label_array)

* annotated ostream insertion operator for full SerialDenseVector with labels

• template<typename OrdinalType, typename ScalarType> void write_data_tabular (std::ostream &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v)

* tabular ostream insertion operator for full SerialDenseVector

• template<typename OrdinalType, typename ScalarType> void write_data_partial_tabular (std::ostream &s, size_t start_index, size_t num_items, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v)

* tabular ostream insertion operator for partial SerialDenseVector

• void write_data_tabular (std::ostream &s, StringMultiArrayConstView ma)

* tabular ostream insertion operator for view of StringMultiArray

• template<typename OrdinalType, typename ScalarType> std::istream & operator>> (std::istream &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &data)

* global std::istream extraction operator for SerialDenseVector

• template<typename OrdinalType, typename ScalarType> std::ostream & operator<<(std::ostream &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &data)

* global std::ostream insertion operator for SerialDenseVector

• template<typename OrdinalType, typename ScalarType> void read_data (BiStream &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, StringMultiArray &label_array)

* SerialDenseVector with labels.

• template<typename OrdinalType, typename ScalarType> void write_data (BoStream &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, const StringMultiArray &label_array)

* SerialDenseVector with labels.

• template<typename OrdinalType, typename ScalarType> void read_data (MPIUnpackBuffer &s, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, StringMultiArray &label_array)

* with labels

• template<typename OrdinalType, typename ScalarType> void write_data (MPIPackBuffer &s, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &v, const StringMultiArray &label_array)

* with labels
• template<typename OrdinalType, typename ScalarType> void read_data (std::istream &s, std::vector<Teuchos::SerialDenseVector<OrdinalType, ScalarType>> &va)
  standard istream extraction operator for std::vector of SerialDenseVectors

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

• template<typename OrdinalType, typename ScalarType> void read_data (BiStream &s, Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &m)
  standard binary stream extraction operator for SerialSymDenseMatrix

• template<typename OrdinalType, typename ScalarType> void read_data (MPIUnpackBuffer &s, Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &m)
  standard MPI buffer extraction operator for SerialSymDenseMatrix

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

• template<typename OrdinalType, typename ScalarType> void write_data (BoStream &s, const Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &m)
  standard binary stream insertion operator for SerialSymDenseMatrix

• template<typename OrdinalType, typename ScalarType> void write_data (MPIPackBuffer &s, const Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &m)
  standard MPI buffer insertion operator for SerialSymDenseMatrix

• template<typename OrdinalType, typename ScalarType> void write_col_vector_trans (std::ostream &s, OrdinalType col, OrdinalType num_items, bool brackets, bool break_line, bool final_rtn, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)
  standard MPI buffer insertion operator for SerialSymDenseMatrix

• template<typename OrdinalType, typename ScalarType> void write_col_vector_trans (std::ostream &s, OrdinalType col, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)
  ostream insertion operator for a column vector from a SerialDenseMatrix

• template<typename OStreamType, typename OrdinalType, typename ScalarType> void write_col_vector_trans (OStreamType &s, OrdinalType col, OrdinalType num_items, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)
  ostream insertion operator for a column vector from a SerialDenseMatrix

• template<typename OStreamType, typename OrdinalType, typename ScalarType> void write_col_vector_trans (OStreamType &s, OrdinalType col, const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)
ostream insertion operator for a column vector from a SerialDenseMatrix

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

istream extraction operator for a column vector from a SerialDenseMatrix

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

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

- bool operator==(const UIntArray &ua, UIntMultiArrayConstView umav)
  equality operator for UIntArray and UIntMultiArrayConstView

- void copy_data(const RealSymMatrix &rsdm, NEWMAT::SymmetricMatrix &sm)
  copy RealSymMatrix to NEWMAT::SymmetricMatrix

- void copy_data(const RealMatrix &rdm, NEWMAT::Matrix &m)
  copy RealMatrix to NEWMAT::Matrix

- void copy_data(const NEWMAT::ColumnVector &cv, RealVector &rdv)
  copy NEWMAT::ColumnVector to RealVector

- void copy_data(const Real *rdv, const int num_items, NEWMAT::ColumnVector &cv)
  copy Real* (column of Teuchos_SerialDenseMatrix) to NEWMAT::ColumnVector

- void copy_data(const RealVector &rdv, NEWMAT::ColumnVector &cv)
  copy RealVector to NEWMAT::ColumnVector

- void copy_data(const DDaceSamplePoint &dsp, RealVector &rdv)
  copy DDACE point to RealVector

- void copy_data(const std::vector<DDaceSamplePoint> &dspa, RealVectorArray &rdva)
  copy DDACE point array to RealVectorArray

- void copy_data(const std::vector<DDaceSamplePoint> &dspa, Real *ptr, const int ptr_len)
  copy DDACE point array to Real*

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

- bool operator!=(const IntArray &dia1, const IntArray &dia2)
  inequality operator for IntArray
- bool operator!=(const ShortArray &dsa1, const ShortArray &dsa2)
  inequality operator for ShortArray

- bool operator!=(const StringArray &dsa1, const StringArray &dsa2)
  inequality operator for StringArray

- bool operator!=(const UIntArray &ua, UIntMultiArrayConstView umav)
  inequality operator for StringArray

- void build_label (String &label, const String &root_label, size_t tag)
  create a label by appending a numerical tag to the root_label

- void build_labels (StringArray &label_array, const String &root_label)
  label_array. Uses build_label().

- void build_labels (StringMultiArray &label_array, const String &root_label)
  label_array. Uses build_label().

- void build_labels_partial (StringArray &label_array, const String &root_label, size_t start_index, size_t num_items)
  of entries in label_array. Uses build_label().

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

- template<class T> void copy_data (const Array<T> &d, T *ptr, const int ptr_len)
  copy Array<T> to T*

- template<typename OrdinalType, typename ScalarType> void copy_data (const Array<Teuchos::SerialDenseVector<OrdinalType, ScalarType>> &sdv, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType>&sdm, OrdinalType nr, OrdinalType nc) 
  copy Teuchos::SerialDenseVector<OT,ST> to Teuchos::SerialDenseMatrix<OT,ST>

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

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

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

- template<class T> void `copy_data` (const std::map<int, T> &im, `Array<T>` &da)
  
  `copy map<int, T>` to `Array<T>` (discard integer keys)

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

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

  `Array<ScalarType>`.

- template<typename OrdinalType, typename ScalarType> void `copy_data` (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv, `Array<Teuchos::SerialDenseVector<OrdinalType, ScalarType>>` &sdva, OrdinalType num_vec, OrdinalType vec_len)

  copy `SerialDenseVector<>` to `Array<SerialDenseVector<>>`.

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

  copy portion of first `SerialDenseVector` to all of second `SerialDenseVector`

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

  copy all of first `SerialDenseVector` to portion of second `SerialDenseVector`
copy portion of first Array\(<T>\) to all of second Array\(<T>\)

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

- template<class T> void copy_data_partial (const Array\(<T>\) &da, boost::multi_array\(<T, 1>\) &bma, size_t start_index_bma)
  
copy all of first Array\(<T>\) to portion of boost::multi_array\(<T, 1>\)

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

- void merge_data_partial (const IntVector &d_array, RealVector &m_array, size_t start_index_ma)
  
aggregate continuous and discrete arrays into a single merged array

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

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

- template<typename OrdinalType, typename ScalarType> void x_y_pairs_to_x_set (const Teuchos::SerialDenseVector< OrdinalType, ScalarType > &xy_pairs, std::set< ScalarType > &x_set)
  
std::set of (x), discarding the y values

- template<typename MultiArrayType, typename DataType> size_t find_index (const MultiArrayType &a, const DataType &search_data)
- template<typename MultiArrayType, typename DakArrayType> void copy_data (const MultiArrayType &ma, DakArrayType &da)
- template<class T> size_t find_index (const boost::multi_array< T, 1 > &bma, const T &search_data)
  
compute the index of an entry within a boost::multi_array

- size_t find_index (UIntMultiArrayConstView bmacv, const unsigned int &search_data)
  
compute the index of an entry within a boost::multi_array view

- size_t find_index (StringMultiArrayConstView bmacv, const String &search_data)
  
compute the index of an entry within a boost::multi_array view

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

- void copy_data (StringMultiArrayConstView ma, StringArray &da)
  
copy boost::multi_array view to Array
- bool data_interface_id_compare (const DataInterface &di, const void *id)
  global comparison function for DataInterface

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

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

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

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

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

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

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

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

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

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

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

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

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

- MPIUnpackBuffer & operator>> (MPIUnpackBuffer &s, DataResponses &data)
  MPIUnpackBuffer extraction operator for DataResponses.
• std::ostream & operator<< (std::ostream &s, const DataResponses &data)
  std::ostream insertion operator for DataResponses

• MPIPackBuffer & operator<< (MPIPackBuffer &s, const DataStrategy &data)
  MPIPackBuffer insertion operator for DataStrategy.

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

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

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

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

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

• std::ostream & operator<< (std::ostream &s, const DataVariables &data)
  std::ostream insertion operator for DataVariables

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

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

• RealVector const * continuous_lower_bounds (Optimizer1 *o)
• RealVector const * continuous_upper_bounds (Optimizer1 *o)
• RealVector const * nonlinear_ineq_constraint_lower_bounds (Optimizer1 *o)
• RealVector const * nonlinear_ineq_constraint_upper_bounds (Optimizer1 *o)
• RealVector const * nonlinear_eq_constraint_targets (Optimizer1 *o)
• RealVector const * linear_ineq_constraint_lower_bounds (Optimizer1 *o)
• RealVector const * linear_ineq_constraint_upper_bounds (Optimizer1 *o)
• RealVector const * linear_eq_constraint_targets (Optimizer1 *o)
• RealMatrix const * linear_eq_constraint_coeffs (Optimizer1 *o)
• void ComputeResponses (Optimizer1 *o, int mode, int n, double *x)
• void GetFuncs (Optimizer1 *o, int m0, int m1, double *f)
• void GetGrads (Optimizer1 *o, int m0, int m1, int n, int js, int js, double *g)
• void GetContVars (Optimizer1 *o, int n, double *x)
• void SetBestContVars (Optimizer1 *o, int n, double *x)
• void SetBestRespFns (Optimizer1 *o, int n, double *x)
• void *dl_constructor (Optimizer1 *, Dakota_funcs *, dl_find_optimum_t *, dl_destructor_t *)
• static RealVector const * continuous_lower_bounds1 (Optimizer1 *o)
• static RealVector const * continuous_upper_bounds1 (Optimizer1 *o)
• static RealVector const * nonlinear_ineq_constraint_lower_bounds1 (Optimizer1 *o)
• static RealVector const * nonlinear_ineq_constraint_upper_bounds1 (Optimizer1 *o)
• static RealVector const * nonlinear_eq_constraint_targets1 (Optimizer1 *o)
• static RealVector const * linear_ineq_constraint_lower_bounds1 (Optimizer1 *o)
• static RealVector const * linear_ineq_constraint_upper_bounds1 (Optimizer1 *o)
• static RealVector const * linear_eq_constraint_targets1 (Optimizer1 *o)
• static RealMatrix const * linear_eq_constraint_coeffs1 (Optimizer1 *o)
• static RealMatrix const * linear_ineq_constraint_coeffs1 (Optimizer1 *o)
• static void ComputeResponses1 (Optimizer1 *o, int mode, int n, double *x)
• static void GetFuncs1 (Optimizer1 *o, int m0, int m1, double *f)
• static void GetGrads1 (Optimizer1 *o, int m0, int m1, int n, int is, int js, double *g)
• static void GetContVars1 (Optimizer1 *o, int n, double *x)
• static void SetBestContVars1 (Optimizer1 *o, int n, double *x)
• static void SetBestDiscVars1 (Optimizer1 *o, int n, int *x)
• static void SetBestRespFns1 (Optimizer1 *o, int n, double *x)
• static double Get_Real1 (Optimizer1 *o, const char *name)
• static int Get_Int1 (Optimizer1 *o, const char *name)
• static bool Get_Bool1 (Optimizer1 *o, const char *name)
• static const char ** arg_adjust (bool cmd_line_args, StringArray &args, const char **av, const char *s)
• static HANDLE * wait_setup (std::map < pid_t, int > *M, size_t *pn)
• static int wait_for_one (size_t n, HANDLE *h, int req1, size_t *pi)
• Real getdist (const RealVector &x1, const RealVector &x2)
• Real mindist (const RealVector &x, const RealMatrix &xset, int except)
• Real mindistindx (const RealVector &x, const RealMatrix &xset, const IntArray &indx)
• Real getRmax (const RealMatrix &xset)
• template<typename T> T abort_handler_t (int code)
• int start_grid_computing (char *analysis_driver_script, char *params_file, char *results_file)
• int stop_grid_computing ()
• int perform_analysis (char *iteration_num)
• template<typename T> string asstring (const T &val)

  Creates a string from the argument val using an ostringstream.

• 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
7.1 Dakota Namespace Reference

- int nitd_parse (const char *, FILE *)
- static void scale_chk (StringArray &ST, RealVector &S, const char *what, const char **univ)
- static void BuildLabels (StringArray *sa, size_t ns, size_t n1, size_t n2, const char **stub)
- static int flist_check (IntList *L, int n, IntArray *iv, const char *what)
- static void flist_check2 (size_t n, IntArray *iv, const char *what)
- static int wronglen (size_t n, RealVector *V, const char *what)
- static int wronglen2 (size_t n, IntVector *V, const char *what)
- static void Vcopyup (RealVector *V, RealVector *M, size_t i, size_t n)
- static void Set_rdv (RealVector *V, double d, size_t n)
- static void Vadj_Normal (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_normalUnc (DataVariablesRep *dv, size_t i0)
- static void Vadj_Lognormal (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_lognormalUnc (DataVariablesRep *dv, size_t i0)
- static void Vadj_Uniform (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_uniformUnc (DataVariablesRep *dv, size_t i0)
- static void Vadj_Logununiform (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_loguniformUnc (DataVariablesRep *dv, size_t i0)
- static void Vadj_Triangular (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_triangularUnc (DataVariablesRep *dv, size_t i0)
- static void Vadj_Exponential (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_Exponential (DataVariablesRep *dv, size_t i0)
- static void Vadj_Beta (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_betaUnc (DataVariablesRep *dv, size_t i0)
- static void Vadj_Gamma (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_Gamma (DataVariablesRep *dv, size_t i0)
- static void Vadj_Gumbel (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_Gumbel (DataVariablesRep *dv, size_t i0)
- static void Vadj_Frechet (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_Frechet (DataVariablesRep *dv, size_t i0)
- static void Vadj_Weibull (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_Weibull (DataVariablesRep *dv, size_t i0)
- static void Vadj_HistogramBin (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vadj_HistogramPt (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_HistogramBin (DataVariablesRep *dv, size_t i0)
- static void Vbgen_HistogramPt (DataVariablesRep *dv, size_t i0)
- static void Vadj_Interval (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vbgen_Interval (DataVariablesRep *dv, size_t i0)
- static void Vadj_Poisson (DataVariablesRep *dv, size_t i0)
- static void Vbgen_Poisson (DataVariablesRep *dv, size_t i0)
- static void Vadj_NegBinomial (DataVariablesRep *dv, size_t i0)
- static void Vbgen_NegBinomial (DataVariablesRep *dv, size_t i0)
- static void Vbgen_Geometric (DataVariablesRep *dv, size_t i0)
- static void Vbgen_HyperGeom (DataVariablesRep *dv, size_t i0)
- static void DIsset (size_t n, IntSetArray *a, IntVector *L, IntVector *U, IntVector *V)
- static void DRset (size_t n, RealSetArray *a, RealVector *L, RealVector *U, RealVector *V)
- static void Vbgen_DDSI (DataVariablesRep *dv, size_t n)
- static void Vbgen_DDSR (DataVariablesRep *dv, size_t n)
- static void Vbgen_DSSI (DataVariablesRep *dv, size_t n)
- static void Vbgen_DSSR (DataVariablesRep *dv, size_t n)
- static void not_div (const char *kind, size_t nsv, size_t m)
- static void wrong_number (const char *what, const char *kind, size_t nsv, size_t m)
- static void too_small (const char *kind)
- static void suppressed (const char *kind, int ndup, int *ip, Real *rp)
- static void bad_initial_ivalue (const char *kind, int val)
- static void bad_initial_rvalue (const char *kind, Real val)
- static void Vadj_DiscreteDesSetReal (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vadj_DiscreteDesSetInt (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vadj_DiscreteStateSetReal (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Vadj_DiscreteStateSetInt (DataVariablesRep *dv, size_t i0, Var_Info *vi)
- static void Rdv_copy (RealVector **prdv, RealVectorArray *rdva)
- static void var_iulbl (const char *keyname, Values *val, VarLabel *vl)
- static Iface_mp_Rlit MP3 (failAction, recoveryFnVals, recover)
- static Iface_mp_ilit MP3 (failAction, retryLimit, retry)
- static Iface_mp_lit MP2 (analysisScheduling, self)
- static Iface_mp_lit MP2 (analysisScheduling, static)
- static Iface_mp_lit MP2 (evalScheduling, self)
- static Iface_mp_lit MP2 (evalScheduling, static)
- static Iface_mp_lit MP2 (failAction, abort)
- static Iface_mp_lit MP2 (failAction, continuation)
- static Iface_mp_lit MP2 (interfaceSynchronization, asynchronous)
- static Iface_mp_lit MP2 (interfaceType, direct)
- static Iface_mp_lit MP2 (interfaceType, fork)
- static Iface_mp_lit MP2 (interfaceType, grid)
- static Iface_mp_lit MP2 (interfaceType, system)
- static Iface_mp_lit MP2 (asynchLocalEvalScheduling, self)
- static Iface_mp_lit MP2 (asynchLocalEvalScheduling, static)
- static bool MP_ (activeSetVectorFlag)
- static bool MP_ (apreproFlag)
- static bool MP_ (dirSave)
- static bool MP_ (dirTag)
- static bool MP_ (evalCacheFlag)
- static bool MP_ (fileSaveFlag)
- static bool MP_ (fileTagFlag)
- static bool MP_ (restartFileFlag)
- static bool MP_ (templateCopy)
- static bool MP_ (templateReplace)
- static bool MP_ (useWorkdir)
- static bool MP_ (verbatimFlag)
- static int MP_ (analysisServers)
- static int MP_ (asynchLocalAnalysisConcurrency)
- static int MP_ (asynchLocalEvalConcurrency)
- static int MP_ (evalServers)
- static int MP_ (procsPerAnalysis)
- static IntVector MP_ (primeBase)
- static IntVector MP_ (sequenceLeap)
- static IntVector MP_ (sequenceStart)
- static IntVector MP_ (stepsPerVariable)
- static Method_mp_ilit2 MP3 (replacementType, numberRetained, chc)
- static Method_mp_ilit2 MP3 (replacementType, numberRetained, elitist)
- static Method_mp_ilit2 MP3 (replacementType, numberRetained, random)
- static Method_mp_ilit2z MP3 (crossoverType, numCrossPoints, multi_point_binary)
- static Method_mp_ilit2z MP3 (crossoverType, numCrossPoints, multi_point_parameterized_binary)
- static Method_mp_ilit2z MP3 (crossoverType, numCrossPoints, multi_point_real)
- static Method_mp_lit MP2 (boxDivision, all_dimensions)
- static Method_mp_lit MP2 (boxDivision, major_dimension)
- static Method_mp_lit MP2 (collocSampleReuse, all)
- static Method_mp_lit MP2 (convergenceType, average_fitness_tracker)
- static Method_mp_lit MP2 (convergenceType, best_fitness_tracker)
- static Method_mp_lit MP2 (convergenceType, metric_tracker)
- static Method_mp_lit MP2 (crossoverType, blend)
- static Method_mp_lit MP2 (crossoverType, two_point)
- static Method_mp_lit MP2 (crossoverType, uniform)
- static Method_mp_lit MP2 (daceMethod, box_behnken)
- static Method_mp_lit MP2 (daceMethod, central_composite)
- static Method_mp_lit MP2 (daceMethod, grid)
- static Method_mp_lit MP2 (daceMethod, lhs)
- static Method_mp_lit MP2 (daceMethod, oas)
- static Method_mp_lit MP2 (daceMethod, random)
- static Method_mp_lit MP2 (distributionType, complementary)
- static Method_mp_lit MP2 (distributionType, cumulative)
- static Method_mp_lit MP2 (evalSynchronization, blocking)
- static Method_mp_lit MP2 (evalSynchronization, nonblocking)
- static Method_mp_lit MP2 (evalSynchronize, blocking)
- static Method_mp_lit MP2 (evalSynchronize, nonblocking)
- static Method_mp_lit MP2 (expansionSampleType, incremental_lhs)
- static Method_mp_lit MP2 (exploratoryMoves, adaptive)
- static Method_mp_lit MP2 (exploratoryMoves, multi_step)
- static Method_mp_lit MP2 (exploratoryMoves, simple)
- static Method_mp_lit MP2 (fitnessType, domination_count)
- static Method_mp_lit MP2 (fitnessType, layer_rank)
- static Method_mp_lit MP2 (fitnessType, linear_rank)
- static Method_mp_lit MP2 (fitnessType, merit_function)
- static Method_mp_lit MP2 (fitnessType, proportional)
- static Method_mp_lit MP2 (initializationType, random)
- static Method_mp_lit MP2 (initializationType, unique_random)
- static Method_mp_lit MP2 (meritFunction, merit_max)
- static Method_mp_lit MP2 (meritFunction, merit_max_smooth)
- static Method_mp_lit MP2 (meritFunction, merit1)
- static Method_mp_lit MP2 (meritFunction, merit1_smooth)
- static Method_mp_lit MP2 (meritFunction, merit2)
- static Method_mp_lit MP2 (meritFunction, merit2_smooth)
- static Method_mp_lit MP2 (meritFunction, merit2_squared)
- static Method_mp_lit MP2 (methodName, asynch_pattern_search)
- static Method_mp_lit MP2 (methodName, coliny_cobyla)
- static Method_mp_lit MP2 (methodName, coliny_direct)
- static Method_mp_lit MP2 (methodName, coliny_pattern_search)
- static Method_mp_lit MP2 (methodName, coliny_solis_wets)
- static Method_mp_lit MP2 (methodName, conmin_frcg)
- static Method_mp_lit MP2 (methodName, conmin_mfd)
- static Method_mp_lit MP2 (methodName, dace)
- static Method_mp_lit MP2 (methodName, dot_bfgs)
- static Method_mp_lit MP2 (methodName, dot_freg)
- static Method_mp_lit MP2 (methodName, dot_mmfd)
- static Method_mp_lit MP2 (methodName, dot_slp)
- static Method_mp_lit MP2 (methodName, dot_sqp)
- static Method_mp_lit MP2 (methodName, efficient_global)
- static Method_mp_lit MP2 (methodName, fsu_cvt)
- static Method_mp_lit MP2 (methodName, fsu_halton)
- static Method_mp_lit MP2 (methodName, fsu_hammersley)
- static Method_mp_lit MP2 (methodName, ncsu_direct)
- static Method_mp_lit MP2 (methodName, nl2sol)
- static Method_mp_lit MP2 (methodName, nlplq1_sqp)
- static Method_mp_lit MP2 (methodName, nlsslq_sqp)
- static Method_mp_lit MP2 (methodName, nond_bayes_calib)
- static Method_mp_lit MP2 (methodName, nond_global_evidence)
- static Method_mp_lit MP2 (methodName, nond_global_interval_est)
- static Method_mp_lit MP2 (methodName, nond_global Reliability)
- static Method_mp_lit MP2 (methodName, nond_local_evidence)
- static Method_mp_lit MP2 (methodName, nond_local_interval_est)
- static Method_mp_lit MP2 (methodName, nond_polynomial_chaos)
- static Method_mp_lit MP2 (methodName, nond_sampling)
- static Method_mp_lit MP2 (methodName, nonlinear_cg)
- static Method_mp_lit MP2 (methodName, npsol_sqp)
- static Method_mp_lit MP2 (methodName, optpp_cg)
- static Method_mp_lit MP2 (methodName, optpp_fd_newton)
- static Method_mp_lit MP2 (methodName, optpp_g_newton)
- static Method_mp_lit MP2 (methodName, optpp_newton)
• static Method_mp_lit MP2 (methodName, optpp_pds)
• static Method_mp_lit MP2 (methodName, optpp_q_newton)
• static Method_mp_lit MP2 (methodName, psuade_moat)
• static Method_mp_lit MP2 (methodName, surrogate_based_global)
• static Method_mp_lit MP2 (methodName, surrogate_based_local)
• static Method_mp_lit MP2 (methodName, vector_parameter_study)
• static Method_mp_lit MP2 (methodName, list_parameter_study)
• static Method_mp_lit MP2 (methodName, centered_parameter_study)
• static Method_mp_lit MP2 (methodName, multidim_parameter_study)
• static Method_mp_lit MP2 (minMaxType, maximize)
• static Method_mp_lit MP2 (minMaxType, minimize)
• static Method_mp_lit MP2 (mutationType, bit_random)
• static Method_mp_lit MP2 (mutationType, offset_cauchy)
• static Method_mp_lit MP2 (mutationType, offset_normal)
• static Method_mp_lit MP2 (mutationType, offset_uniform)
• static Method_mp_lit MP2 (mutationType, replace_uniform)
• static Method_mp_lit MP2 (nondOptAlgorithm, nip)
• static Method_mp_lit MP2 (nondOptAlgorithm, sqp)
• static Method_mp_lit MP2 (nondOptAlgorithm, lhs)
• static Method_mp_lit MP2 (nondOptAlgorithm, ego)
• static Method_mp_lit MP2 (patternBasis, coordinate)
• static Method_mp_lit MP2 (patternBasis, simplex)
• static Method_mp_lit MP2 (reliabilityIntegration, first_order)
• static Method_mp_lit MP2 (reliabilityIntegration, second_order)
• static Method_mp_lit MP2 (reliabilityIntegrationRefine, ais)
• static Method_mp_lit MP2 (reliabilityIntegrationRefine, is)
• static Method_mp_lit MP2 (reliabilityIntegrationRefine, mmais)
• static Method_mp_lit MP2 (reliabilitySearchType, amv_plus_u)
• static Method_mp_lit MP2 (reliabilitySearchType, amv_plus_x)
• static Method_mp_lit MP2 (reliabilitySearchType, amv_u)
• static Method_mp_lit MP2 (reliabilitySearchType, amv_x)
• static Method_mp_lit MP2 (reliabilitySearchType, egra_u)
• static Method_mp_lit MP2 (reliabilitySearchType, egra_x)
• static Method_mp_lit MP2 (reliabilitySearchType, no_approx)
• static Method_mp_lit MP2 (reliabilitySearchType, tana_u)
• static Method_mp_lit MP2 (reliabilitySearchType, tana_x)
• static Method_mp_lit MP2 (replacementType, elitist)
• static Method_mp_lit MP2 (replacementType, favor_feasible)
• static Method_mp_lit MP2 (replacementType, roulette_wheel)
• static Method_mp_lit MP2 (replacementType, unique_roulette_wheel)
• static Method_mp_lit MP2 (responseLevelMappingType, gen_reliabilities)
• static Method_mp_lit MP2 (responseLevelMappingType, probabilities)
• static Method_mp_lit MP2 (responseLevelMappingType, reliabilities)
• static Method_mp_lit MP2 (rngName, mt19937)
• static Method_mp_lit MP2 (rngName, rnum2)
• static Method_mp_lit MP2 (sampleType, incremental_lhs)
• static Method_mp_lit MP2 (sampleType, incremental_random)
• static Method_mp_lit MP2 (sampleType, lhs)
• static Method_mp_lit MP2 (sampleType, random)
• static Method_mp_lit MP2 (searchMethod, gradient_based_line_search)
• static Method_mp_lit MP2 (searchMethod, tr_pds)
• static Method_mp_lit MP2 (searchMethod, trust_region)
• static Method_mp_lit MP2 (searchMethod, value_based_line_search)
• static Method_mp_lit MP2 (trialType, grid)
• static Method_mp_lit MP2 (trialType, halton)
• static Method_mp_lit MP2 (trialType, random)
• static Method_mp_lit2 MP4 (methodName, reliabilitySearchType, nond_local_reliability,"mv")
• static Method_mp_litc MP3 (crossoverType, crossoverRate, shuffle_random)
• static Method_mp_litc MP3 (crossoverType, crossoverRate, null_crossover)
• static Method_mp_litc MP3 (mutationType, mutationRate, null_mutation)
• static Method_mp_litc MP3 (mutationType, mutationRate, offset_cauchy)
• static Method_mp_litc MP3 (mutationType, mutationRate, offset_normal)
• static Method_mp_litc MP3 (mutationType, mutationRate, offset_uniform)
• static Method_mp_litc MP3 (replacementType, fitnessLimit, below_limit)
• static Method_mp_litr MP3 (nichingType, nicheVector, distance)
• static Method_mp_litr MP3 (nichingType, nicheVector, radial)
• static Method_mp_litr MP3 (postProcessorType, distanceVector, distance_postprocessor)
• static Method_mp_slit MP2a (methodOutput, DEBUG_OUTPUT)
• static Method_mp_slit MP2a (methodOutput, QUIET_OUTPUT)
• static Method_mp_slit MP2a (methodOutput, SILENT_OUTPUT)
• static Method_mp_slit MP2a (methodOutput, VERBOSE_OUTPUT)
• static Method_mp_slit MP2a (surrBasedLocalAcceptLogic, FILTER)
• static Method_mp_slit MP2a (surrBasedLocalAcceptLogic, TR_RATIO)
• static Method_mp_slit MP2a (surrBasedLocalConstrRelax, HOMOTOPY)
• static Method_mp_slit MP2a (surrBasedLocalConstrRelax, ADAPTIVE_PENALTY_MERIT)
• static Method_mp_slit MP2a (surrBasedLocalConstrRelax, AUGMENTED_LAGRANGIAN_MERIT)
• static Method_mp_slit MP2a (surrBasedLocalConstrRelax, LAGRANGIAN_MERIT)
• static Method_mp_slit MP2a (surrBasedLocalConstrRelax, PENALTY_MERIT)
• static Method_mp_slit MP2a (surrBasedLocalSubProbCon, LINEARIZED_CONSTRAINTS)
• static Method_mp_slit MP2a (surrBasedLocalSubProbCon, NO_CONSTRAINTS)
• static Method_mp_slit MP2a (surrBasedLocalSubProbCon, ORIGINAL_CONSTRAINTS)
• static Method_mp_slit MP2a (surrBasedLocalSubProbObj, AUGMENTED_LAGRANGIAN_OBJECTIVE)
• static Method_mp_slit MP2a (surrBasedLocalSubProbObj, LAGRANGIAN_OBJECTIVE)
• static Method_mp_slit MP2a (surrBasedLocalSubProbObj, ORIGINAL_PRIMARY)
• static Method_mp_slit MP2a (surrBasedLocalSubProbObj, SINGLE_OBJECTIVE)
• static Method_mp_slit2 MP3 (initializationType, flatFile, flat_file)
• static Method_mp_slit2 MP3 (methodName, dlDetails, dl_solver)
• static Real MP_ (absConvTol)
• static Real MP_ (centeringParam)
• static Real MP_ (collocationRatio)
• static Real MP_ (constraintPenalty)
• static Real MP_(constrPenalty)
• static Real MP_(constraintTolerance)
• static Real MP_(contractFactor)
• static Real MP_(contractStepLength)
• static Real MP_(convergenceTolerance)
• static Real MP_(crossoverRate)
• static Real MP_(falseConvTol)
• static Real MP_(functionPrecision)
• static Real MP_(globalBalanceParam)
• static Real MP_(gradientTolerance)
• static Real MP_(initDelta)
• static Real MP_(initStepLength)
• static Real MP_(initTRRadius)
• static Real MP_(lineSearchTolerance)
• static Real MP_(localBalanceParam)
• static Real MP_(maxBoxSize)
• static Real MP_(maxStep)
• static Real MP_(minBoxSize)
• static Real MP_(mutationRate)
• static Real MP_(mutationScale)
• static Real MP_(shrinkagePercent)
• static Real MP_(singConvTol)
• static Real MP_(singRadius)
• static Real MP_(smoothFactor)
• static Real MP_(solnTarget)
• static Real MP_(stepLenToBoundary)
• static Real MP_(surrBasedLocalTRContract)
• static Real MP_(surrBasedLocalTRContractTrigger)
• static Real MP_(surrBasedLocalTRExpand)
• static Real MP_(surrBasedLocalTRExpandTrigger)
• static Real MP_(surrBasedLocalTRInitSize)
• static Real MP_(surrBasedLocalTRMinSize)
• static Real MP_(threshDelta)
• static Real MP_(threshStepLength)
• static Real MP_(volBoxSize)
• static Real MP_(xConvTol)
• static RealVector MP_(finalPoint)
• static RealVector MP_(linearEqConstraintCoeffs)
• static RealVector MP_(linearEqTargets)
• static RealVector MP_(linearEqScales)
• static RealVector MP_(linearIneqConstraintCoeffs)
• static RealVector MP_(linearIneqLowerBnds)
• static RealVector MP_(linearIneqUpperBnds)
• static RealVector MP_(listOfPoints)
• static RealVector MP_(sparseGridDimPref)
- static RealVector MP_ (stepVector)
- static unsigned short MP_ (sparseGridLevel)
- static bool MP_ (allVarsFlag)
- static bool MP_ (constantPenalty)
- static bool MP_ (expansionFlag)
- static bool MP_ (fixedSeedFlag)
- static bool MP_ (fixedSequenceFlag)
- static bool MP_ (latinizeFlag)
- static bool MP_ (mainEffectsFlag)
- static bool MP_ (methodScaling)
- static bool MP_ (mutationAdaptive)
- static bool MP_ (printPopFlag)
- static bool MP_ (randomizeOrderFlag)
- static bool MP_ (regressDiag)
- static bool MP_ (showMiscOptions)
- static bool MP_ (speculativeFlag)
- static bool MP_ (surrBasedGlobalReplacePts)
- static bool MP_ (surrBasedLocalLayerBypass)
- static bool MP_ (varBasedDecompFlag)
- static bool MP_ (volQualityFlag)
- static short MP_ (expansionType)
- static int MP_ (collocationPoints)
- static int MP_ (contractAfterFail)
- static int MP_ (covarianceType)
- static int MP_ (expandAfterSuccess)
- static int MP_ (expansionSamples)
- static int MP_ (expansionTerms)
- static int MP_ (maxFunctionEvaluations)
- static int MP_ (maxIterations)
- static int MP_ (mutationRange)
- static int MP_ (newSolnsGenerated)
- static int MP_ (numSamples)
- static int MP_ (numSteps)
- static int MP_ (numSymbols)
- static int MP_ (numTrials)
- static int MP_ (populationSize)
- static int MP_ (previousSamples)
- static int MP_ (randomSeed)
- static int MP_ (searchSchemeSize)
- static int MP_ (surrBasedLocalSoftConvLimit)
- static int MP_ (totalPatternSize)
- static int MP_ (verifyLevel)
- static size_t MP_ (numGenerations)
- static size_t MP_ (numOffspring)
- static size_t MP_ (numParents)
- static Method_mp_type MP2s (expansionType, ASKEY_U)
- static Method_mp_type MP2s (expansionType, STD_NORMAL_U)
- static IntSet MP_ (surrogateFnIndices)
- static Model_mp_lit MP2 (approxCorrectionType, additive)
- static Model_mp_lit MP2 (approxCorrectionType, combined)
- static Model_mp_lit MP2 (approxCorrectionType, multiplicative)
- static Model_mp_lit MP2 (approxSampleReuse, all)
- static Model_mp_lit MP2 (approxSampleReuse, none)
- static Model_mp_lit MP2 (approxSampleReuse, region)
- static Model_mp_lit MP2 (marsInterpolation, linear)
- static Model_mp_lit MP2 (marsInterpolation, cubic)
- static Model_mp_lit MP2 (modelType, nested)
- static Model_mp_lit MP2 (modelType, single)
- static Model_mp_lit MP2 (modelType, surrogate)
- static Model_mp_lit MP2 (surrogateType, hierarchical)
- static Model_mp_lit MP2 (surrogateType, global_gaussian)
- static Model_mp_lit MP2 (surrogateType, global_kriging)
- static Model_mp_lit MP2 (surrogateType, global_mars)
- static Model_mp_lit MP2 (surrogateType, global_moving_least_squares)
- static Model_mp_lit MP2 (surrogateType, global_neural_network)
- static Model_mp_lit MP2 (surrogateType, global_polynomial)
- static Model_mp_lit MP2 (surrogateType, global_radial_basis)
- static Model_mp_lit MP2 (surrogateType, local_taylor)
- static Model_mp_lit MP2 (surrogateType, multipoint_tana)
- static Model_mp_ord MP2s (approxCorrectionOrder, 0)
- static Model_mp_ord MP2s (approxCorrectionOrder, 1)
- static Model_mp_ord MP2s (approxCorrectionOrder, 2)
- static Model_mp_ord MP2s (polynomialOrder, 1)
- static Model_mp_ord MP2s (polynomialOrder, 2)
- static Model_mp_ord MP2s (polynomialOrder, 3)
- static Model_mp_ord MP2s (trendOrder, 0)
- static Model_mp_ord MP2s (trendOrder, 1)
- static Model_mp_ord MP2s (trendOrder, 2)
- static Real MP_ (annRange)
- static RealVector MP_ (krigingConminSeed)
- static RealVector MP_ (krigingCorrelations)
- static RealVector MP_ (krigingMaxCorrelations)
- static RealVector MP_ (krigingMinCorrelations)
- static RealVector MP_ (primaryRespCoeffs)
- static RealVector MP_ (secondaryRespCoeffs)
- static bool MP_ (approxGradUsageFlag)
- static bool MP_ (pointSelection)
- static short MP_ (annNodes)
- static short MP_ (annRandomWeight)
- static short MP_ (krigingMaxTrials)
- static short MP_ (marsMaxBases)
- static short MP_ (mlsPolyOrder)
- static short MP_ (mlsWeightFunction)
- static short MP_ (rbfBases)
- static short MP_ (rbfMaxPts)
- static short MP_ (rbfMaxSubsets)
- static short MP_ (rbfMinPartition)
- static RealVector MP_ (primaryRespFnWeights)
- static RealVector MP_ (nonlinearEqTargets)
- static RealVector MP_ (nonlinearIneqLowerBnds)
- static RealVector MP_ (nonlinearIneqUpperBnds)
- static RealVector MP_ (primaryRespFnScales)
- static RealVector MP_ (nonlinearEqScales)
- static RealVector MP_ (nonlinearIneqScales)
- static Resp_mp_lit MP2 (gradientType, analytic)
- static Resp_mp_lit MP2 (gradientType, mixed)
- static Resp_mp_lit MP2 (gradientType, none)
- static Resp_mp_lit MP2 (gradientType, numerical)
- static Resp_mp_lit MP2 (hessianType, analytic)
- static Resp_mp_lit MP2 (hessianType, mixed)
- static Resp_mp_lit MP2 (hessianType, none)
- static Resp_mp_lit MP2 (hessianType, numerical)
- static Resp_mp_lit MP2 (hessianType, quasi)
- static Resp_mp_lit MP2 (intervalType, central)
- static Resp_mp_lit MP2 (intervalType, forward)
- static Resp_mp_lit MP2 (methodSource, dakota)
- static Resp_mp_lit MP2 (methodSource, vendor)
- static Resp_mp_lit MP2 (quasiHessianType, bfgs)
- static Resp_mp_lit MP2 (quasiHessianType, damped_bfgs)
- static Resp_mp_lit MP2 (quasiHessianType, sr1)
- static bool MP_ (centralHess)
- static bool MP_ (ignoreBounds)
- static size_t MP_ (numLeastSqTerms)
- static size_t MP_ (numNonlinearEqConstraints)
- static size_t MP_ (numNonlinearIneqConstraints)
- static size_t MP_ (numObjectiveFunctions)
- static size_t MP_ (numResponseFunctions)
- static Real MP_ (hybridLSProb)
- static RealVector MP_ (concurrentParameterSets)
- static Strategy_mp_lit MP2 (hybridType, collaborative)
- static Strategy_mp_lit MP2 (hybridType, embedded)
- static Strategy_mp_lit MP2 (hybridType, sequential)
- static Strategy_mp_lit MP2 (iteratorScheduling, self)
- static Strategy_mp_lit MP2 (iteratorScheduling, static)
- static Strategy_mp_lit MP2 (strategyType, hybrid)
- static Strategy_mp_lit MP2 (strategyType, multi_start)
- static Strategy_mp_lit MP2 (strategyType, pareto_set)
- static Strategy_mp_lit MP2 (strategyType, single_method)
- static bool MP_ (graphicsFlag)
- static bool MP_ (tabularDataFlag)
- static int MP_ (concurrentRandomJobs)
- static int MP_ (concurrentSeed)
- static int MP_ (iteratorServers)
- static size_t MP_ (hybridNumSolnsTrans)
- static size_t MP_ (numBetaUncVars)
- static size_t MP_ (numBinomialUncVars)
- static size_t MP_ (numContinuousDesVars)
- static size_t MP_ (numContinuousStateVars)
- static size_t MP_ (numDiscreteDesRangeVars)
- static size_t MP_ (numDiscreteDesSetIntVars)
- static size_t MP_ (numDiscreteDesSetRealVars)
- static size_t MP_ (numExponentialUncVars)
- static size_t MP_ (numFrechetUncVars)
- static size_t MP_ (numGammaUncVars)
- static size_t MP_ (numGeometricUncVars)
- static size_t MP_ (numGumbelUncVars)
- static size_t MP_ (numHistogramBinUncVars)
- static size_t MP_ (numHistogramPtUncVars)
- static size_t MP_ (numHyperGeomUncVars)
- static size_t MP_ (numHypergeomUnc_trials)
- static size_t MP_ (numIntervalUncVars)
- static size_t MP_ (numLognormalUncVars)
- static size_t MP_ (numLoguniformUncVars)
- static size_t MP_ (numNegBinomialUncVars)
- static size_t MP_ (numNormalUncVars)
- static size_t MP_ (numPoissonUncVars)
- static size_t MP_ (numTriangularUncVars)
- static size_t MP_ (numUniformUncVars)
- static size_t MP_ (numWeibullUncVars)
- static IntVector MP_ (binomialUncNumTrials)
- static IntVector MP_ (hyperGeomUncTotalPop)
- static IntVector MP_ (hyperGeomUncSelectedPop)
- static IntVector MP_ (hyperGeomUncNumDrawn)
- static IntVector MP_ (negBinomialUncNumTrials)
- static IntVector MP_ (discreteDesignRangeLowerBnds)
- static IntVector MP_ (discreteDesignRangeUpperBnds)
- static IntVector MP_ (discreteDesignRangeVars)
- static IntVector MP_ (discreteDesignSetIntVars)
- static IntVector MP_ (discreteStateRangeLowerBnds)
- static IntVector MP_ (discreteStateRangeUpperBnds)
- static IntVector MP_ (discreteStateRangeVars)
- static IntVector MP_ (discreteStateSetIntVars)
- static RealVector MP_ (betaUncLowerBnds)
- static RealVector MP_ (betaUncUpperBnds)
- static RealVector MP_ (binomialUncProbPerTrial)
- static RealVector MP_ (continuousDesignLowerBnds)
- static RealVector MP_ (continuousDesignUpperBnds)
- static RealVector MP_ (continuousDesignVars)
- static RealVector MP_ (continuousDesignScales)
- static RealVector MP_ (discreteDesignSetRealVars)
- static RealVector MP_ (discreteStateSetRealVars)
- static RealVector MP_ (frechetUncBetas)
- static RealVector MP_ (geometricUncProbPerTrial)
- static RealVector MP_ (gumbelUncBetas)
- static RealVector MP_ (negBinomialUncProbPerTrial)
- static RealVector MP_ (normalUncLowerBnds)
- static RealVector MP_ (normalUncMeans)
- static RealVector MP_ (normalUncUpperBnds)
- static RealVector MP_ (poissonUncLambdas)
- static RealVector MP_ (triangularUncModes)
- static RealVector VP_ (dsvr)
- static RealVector VP_ (Ivb)
- static RealVector VP_ (Ivp)
- static RealVector VP_ (ba)
- static RealVector VP_ (bo)
- static RealVector VP_ (bc)
- static RealVector VP_ (pa)
- static RealVector VP_ (pc)
- static RealVector VP_ (ucm)
- static RealVector VP_ (ssvr)
- static Var_brv MP2s (betaUncAlphas, 0.)
- static Var_brv MP2s (betaUncBetas, 0.)
- static Var_brv MP2s (exponentialUncBetas, 0.)
- static Var_brv MP2s (frechetUncAlphas, 2.)
- static Var_brv MP2s (gammaUncAlphas, 0.)
- static Var_brv MP2s (gammaUncBetas, 0.)
- static Var_brv MP2s (gumbelUncAlphas, 0.)
- static Var_brv MP2s (lognormalUncErrFacts, 1.)
- static Var_brv MP2s (lognormalUncLambdas, 0.)
- static Var_brv MP2s (lognormalUncLowerBnds, 0.)
- static Var_brv MP2s (lognormalUncMeans, 0.)
- static Var_brv MP2s (lognormalUncStdDevs, 0.)
- static Var_brv MP2s (lognormalUncUpperBnds, DBL_MAX)
7.1 Dakota Namespace Reference

- static Var_brpv MP2s (lognormalUncZetas, 0.)
- static Var_brpv MP2s (loguniformUncLowerBnds, 0.)
- static Var_brpv MP2s (loguniformUncUpperBnds, DBL_MAX)
- static Var_brpv MP2s (normalUncStdDevs, 0.)
- static Var_brpv MP2s (triangularUncLowerBnds, DBL_MAX)
- static Var_brpv MP2s (triangularUncUpperBnds, DBL_MAX)
- static Var_brpv MP2s (uniformUncLowerBnds, DBL_MAX)
- static Var_brpv MP2s (uniformUncUpperBnds, DBL_MAX)
- static Var_brpv MP2s (weibullUncAlphas, 0.)
- static Var_brpv MP2s (weibullUncBetas, 0.)
- static const char ∗ Var_Name (StringArray ∗sa, char ∗buf, size_t i)
- void dn2f (int ∗n, int ∗p, Real ∗x, Calcrj, int ∗iv, int ∗lv, Real ∗v, int ∗ui, void ∗ur, Vf)
- void dn2fb (int ∗n, int ∗p, Real ∗x, Real ∗b, Calcrj, int ∗iv, int ∗lv, Real ∗v, int ∗ui, void ∗ur, Vf)
- void dn2g (int ∗n, int ∗p, Real ∗x, Calcrj, Calcrj, int ∗iv, int ∗lv, Real ∗v, int ∗ui, void ∗ur, Vf)
- void dn2gb (int ∗n, int ∗p, Real ∗x, Real ∗b, Calcrj, Calcrj, int ∗iv, int ∗lv, Real ∗v, int ∗ui, void ∗ur, Vf)
- void divset_ (int ∗, int ∗, int ∗, int ∗, Real ∗)
- double dr7mdc_ (int ∗)
- static void Rswapchk (Nl2Misc ∗q)
- static int hasnaninf (const double ∗d, int n)
- Real rel_change_c_star (const RealVector &curr_c_star, const RealVector &prev_c_star)
  Computes relative change between successive c_stars using Euclidean norm.

- std::istream & operator>>(std::istream &s, ParamResponsePair &pair)
  std::istream extraction operator for ParamResponsePair

- std::ostream & operator<<(std::ostream &s, const ParamResponsePair &pair)
  std::ostream insertion operator for ParamResponsePair

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

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

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

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

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

- bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  inequality operator
• static void *binsearch (void *kw, size_t kwsize, size_t n, const char *key)
• static const char *Begins (const String &entry_name, const char *s)
• static void Bad_name (String entry_name, const char *where)
• static void Locked_db ()
• static void Null_rep (const char *who)
• static void Null_rep1 (const char *who)
• bool set_compare (const ParamResponsePair &database_pr, const ActiveSet &search_set)
  on ActiveSet content (request vector and derivative variables vector)
• bool id_vars_exact_compare (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr)
  search function for a particular ParamResponsePair within a PRPMultiIndex
• std::size_t hash_value (const ParamResponsePair &prp)
  hash_value for ParamResponsePairs stored in a PRPMultiIndex
• PRPCacheHIter hashedCacheBegin (PRPCache &prp_cache)
  hashed definition of cache begin
• PRPCacheHIter hashedCacheEnd (PRPCache &prp_cache)
  hashed definition of cache end
• PRPQueueHIter hashedQueueBegin (PRPQueue &prp_queue)
  hashed definition of queue begin
• PRPQueueHIter hashedQueueEnd (PRPQueue &prp_queue)
  hashed definition of queue end
• PRPCacheHIter lookup_by_val (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr)
  ActiveSet search data within search_pr.
• bool lookup_by_val (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr, ParamResponsePair &found_pr)
  lookup_by_val(PRPMultiIndexCache&, ParamResponsePair&)
• PRPCacheHIter lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)
  based on interface id, variables, and ActiveSet search data
• bool lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, ParamResponsePair &found_pr)
  variables, and ActiveSet search data
• bool lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, int &found_eval_id)
7.1 Dakota Namespace Reference

based on interface id, variables, and ActiveSet search data

- bool lookup_by_val (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, Response &found_resp)

PRPCacheOIter lookup_by_ids (PRPMultiIndexCache &prp_cache, const IntStringPair &search_ids)

(i.e. std::pair<eval_id,interface_id>) search data

- bool lookup_by_ids (PRPMultiIndexCache &prp_cache, const IntStringPair &search_eval_interface_ids, ParamResponsePair &found_pr)

- bool lookup_by_ids (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr, ParamResponsePair &found_pr)

PRPQueueHIter lookup_by_val (PRPMultiIndexQueue &prp_queue, const ParamResponsePair &search_pr)

ActiveSet search data within search_pr.

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const ParamResponsePair &search_pr, ParamResponsePair &found_pr)

PRPQueueHIter lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)

based on interface id, variables, and ActiveSet search data

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, ParamResponsePair &found_pr)

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, int &found_eval_id)

based on interface id, variables, and ActiveSet search data

- bool lookup_by_val (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, Response &found_resp)

based on interface id, variables, and ActiveSet search data

- PRPQueueOIter lookup_by_eval_id (PRPMultiIndexQueue &prp_queue, const int &search_id)

(i.e. integer eval_id) search data

- bool lookup_by_eval_id (PRPMultiIndexQueue &prp_queue, const int &search_id, ParamResponsePair &found_pr)

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

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

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

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

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

• void concatenate_restart (int argc, char **argv)
  
  concatenate multiple restart files

Variables

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

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

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

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

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

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

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

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

• `const char * FIELD_NAMES [ ]`
• `const int NUMBER_OF_FIELDS = 25`
• **PRPCache data_pairs**
  
  contains all parameter/response pairs.

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

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

• **PRPCache data_pairs**
  
  contains all parameter/response pairs.

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

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

• `int write_precision = 10`
  
  used in ostream data output functions (*restart_util.C* overrides this default value).

• **ParallelLibrary dummy_lib (0)**
  
  *dummy ParallelLibrary* object used for mandatory reference initialization when a real *ParallelLibrary* instance is unavailable.

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

• `int mc_ptr_int = 0`
  
  global pointer for ModelCenter API.
• int dc_ptr_int = 0
  global pointer for ModelCenter eval DB

• ProblemDescDB * Dak_pddb
  set by main(), for use in abort_handler()

• const size_t _NPOS = ~(size_t)0
  special value returned by index() when entry not found

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

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

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

• int mc_ptr_int
  global pointer for ModelCenter API

• int dc_ptr_int
  global pointer for ModelCenter eval DB

• static GuiKeyWord kw_1 [3]
• static GuiKeyWord kw_2 [1]
• static GuiKeyWord kw_3 [4]
• static GuiKeyWord kw_4 [2]
• static GuiKeyWord kw_5 [7]
• static GuiKeyWord kw_6 [7]
• static GuiKeyWord kw_7 [9]
• static GuiKeyWord kw_8 [4]
• static GuiKeyWord kw_9 [10]
• static GuiKeyWord kw_10 [7]
• static GuiKeyWord kw_11 [2]
• static GuiKeyWord kw_12 [18]
• static GuiKeyWord kw_13 [3]
• static GuiKeyWord kw_14 [16]
• static GuiKeyWord kw_15 [2]
• static GuiKeyWord kw_16 [20]
• static GuiKeyWord kw_17 [3]
• static GuiKeyWord kw_18 [2]
• static GuiKeyWord kw_19 [3]
• static GuiKeyWord kw_20 [2]
• static GuiKeyWord kw_21 [5]
• static GuiKeyWord kw_22 [4]
• static GuiKeyWord kw_23 [23]
• static GuiKeyWord kw_24 [3]
• static GuiKeyWord kw_25 [2]
• static GuiKeyWord kw_26 [2]
• static GuiKeyWord kw_27 [26]
• static GuiKeyWord kw_28 [22]
• static GuiKeyWord kw_29 [9]
• static GuiKeyWord kw_30 [14]
• static GuiKeyWord kw_31 [2]
• static GuiKeyWord kw_32 [10]
• static GuiKeyWord kw_33 [1]
• static GuiKeyWord kw_34 [3]
• static GuiKeyWord kw_35 [8]
• static GuiKeyWord kw_36 [10]
• static GuiKeyWord kw_37 [1]
• static GuiKeyWord kw_38 [2]
• static GuiKeyWord kw_39 [5]
• static GuiKeyWord kw_40 [3]
• static GuiKeyWord kw_41 [1]
• static GuiKeyWord kw_42 [6]
• static GuiKeyWord kw_43 [3]
• static GuiKeyWord kw_44 [2]
• static GuiKeyWord kw_45 [2]
• static GuiKeyWord kw_46 [1]
• static GuiKeyWord kw_47 [2]
• static GuiKeyWord kw_48 [4]
• static GuiKeyWord kw_49 [12]
• static GuiKeyWord kw_50 [1]
• static GuiKeyWord kw_51 [4]
• static GuiKeyWord kw_52 [9]
• static GuiKeyWord kw_53 [2]
• static GuiKeyWord kw_54 [3]
• static GuiKeyWord kw_55 [2]
• static GuiKeyWord kw_56 [1]
• static GuiKeyWord kw_57 [1]
• static GuiKeyWord kw_58 [2]
• static GuiKeyWord kw_59 [2]
• static GuiKeyWord kw_60 [2]
• static GuiKeyWord kw_61 [9]
• static GuiKeyWord kw_62 [2]
• static GuiKeyWord kw_63 [5]
• static GuiKeyWord kw_64 [2]
• static GuiKeyWord kw_65 [1]
• static GuiKeyWord kw_66 [1]
• static GuiKeyWord kw_67 [2]
• static GuiKeyWord kw_68 [2]
• static GuiKeyWord kw_69 [2]
• static GuiKeyWord kw_70 [9]
• static GuiKeyWord kw_71 [2]
• static GuiKeyWord kw_72 [2]
• static GuiKeyWord kw_73 [7]
• static GuiKeyWord kw_74 [2]
• static GuiKeyWord kw_75 [1]
• static GuiKeyWord kw_76 [1]
• static GuiKeyWord kw_77 [2]
• static GuiKeyWord kw_78 [2]
• static GuiKeyWord kw_79 [6]
• static GuiKeyWord kw_80 [2]
• static GuiKeyWord kw_81 [5]
• static GuiKeyWord kw_82 [3]
• static GuiKeyWord kw_83 [9]
• static GuiKeyWord kw_84 [1]
• static GuiKeyWord kw_85 [3]
• static GuiKeyWord kw_86 [2]
• static GuiKeyWord kw_87 [7]
• static GuiKeyWord kw_88 [1]
• static GuiKeyWord kw_89 [3]
• static GuiKeyWord kw_90 [2]
• static GuiKeyWord kw_91 [3]
• static GuiKeyWord kw_92 [2]
• static GuiKeyWord kw_93 [3]
• static GuiKeyWord kw_94 [2]
• static GuiKeyWord kw_95 [1]
• static GuiKeyWord kw_96 [19]
• static GuiKeyWord kw_97 [1]
• static GuiKeyWord kw_98 [4]
• static GuiKeyWord kw_99 [12]
• static GuiKeyWord kw_100 [2]
• static GuiKeyWord kw_101 [1]
• static GuiKeyWord kw_102 [15]
• static GuiKeyWord kw_103 [1]
• static GuiKeyWord kw_104 [12]
• static GuiKeyWord kw_105 [11]
• static GuiKeyWord kw_106 [10]
• static GuiKeyWord kw_107 [4]
• static GuiKeyWord kw_108 [16]
• static GuiKeyWord kw_109 [4]
• static GuiKeyWord kw_110 [3]
• static GuiKeyWord kw_111 [2]
• static GuiKeyWord kw_112 [2]
• static GuiKeyWord kw_113 [2]
- static GuiKeyWord kw_114 [2]
- static GuiKeyWord kw_115 [4]
- static GuiKeyWord kw_116 [10]
- static GuiKeyWord kw_117 [3]
- static GuiKeyWord kw_118 [2]
- static GuiKeyWord kw_119 [7]
- static GuiKeyWord kw_120 [1]
- static GuiKeyWord kw_121 [4]
- static GuiKeyWord kw_122 [6]
- static GuiKeyWord kw_123 [18]
- static GuiKeyWord kw_124 [3]
- static GuiKeyWord kw_125 [60]
- static GuiKeyWord kw_126 [1]
- static GuiKeyWord kw_127 [4]
- static GuiKeyWord kw_128 [2]
- static GuiKeyWord kw_129 [1]
- static GuiKeyWord kw_130 [6]
- static GuiKeyWord kw_131 [3]
- static GuiKeyWord kw_132 [2]
- static GuiKeyWord kw_133 [5]
- static GuiKeyWord kw_134 [2]
- static GuiKeyWord kw_135 [2]
- static GuiKeyWord kw_136 [2]
- static GuiKeyWord kw_137 [3]
- static GuiKeyWord kw_138 [3]
- static GuiKeyWord kw_139 [4]
- static GuiKeyWord kw_140 [3]
- static GuiKeyWord kw_141 [13]
- static GuiKeyWord kw_142 [6]
- static GuiKeyWord kw_143 [3]
- static GuiKeyWord kw_144 [2]
- static GuiKeyWord kw_145 [2]
- static GuiKeyWord kw_146 [5]
- static GuiKeyWord kw_147 [6]
- static GuiKeyWord kw_148 [1]
- static GuiKeyWord kw_149 [10]
- static GuiKeyWord kw_150 [2]
- static GuiKeyWord kw_151 [1]
- static GuiKeyWord kw_152 [2]
- static GuiKeyWord kw_153 [5]
- static GuiKeyWord kw_154 [3]
- static GuiKeyWord kw_155 [4]
- static GuiKeyWord kw_156 [6]
- static GuiKeyWord kw_157 [3]
- static GuiKeyWord kw_158 [4]
- static GuiKeyWord kw_159 [5]
- static GuiKeyWord kw_160 [8]
- static GuiKeyWord kw_161 [4]
- static GuiKeyWord kw_162 [1]
- static GuiKeyWord kw_163 [2]
- static GuiKeyWord kw_164 [15]
- static GuiKeyWord kw_165 [1]
- static GuiKeyWord kw_166 [3]
- static GuiKeyWord kw_167 [2]
- static GuiKeyWord kw_168 [5]
- static GuiKeyWord kw_169 [1]
- static GuiKeyWord kw_170 [3]
- static GuiKeyWord kw_171 [1]
- static GuiKeyWord kw_172 [5]
- static GuiKeyWord kw_173 [1]
- static GuiKeyWord kw_174 [1]
- static GuiKeyWord kw_175 [9]
- static GuiKeyWord kw_176 [10]
- static GuiKeyWord kw_177 [3]
- static GuiKeyWord kw_178 [12]
- static GuiKeyWord kw_179 [8]
- static GuiKeyWord kw_180 [8]
- static GuiKeyWord kw_181 [4]
- static GuiKeyWord kw_182 [4]
- static GuiKeyWord kw_183 [8]
- static GuiKeyWord kw_184 [4]
- static GuiKeyWord kw_185 [4]
- static GuiKeyWord kw_186 [4]
- static GuiKeyWord kw_187 [6]
- static GuiKeyWord kw_188 [6]
- static GuiKeyWord kw_189 [2]
- static GuiKeyWord kw_190 [6]
- static GuiKeyWord kw_191 [10]
- static GuiKeyWord kw_192 [8]
- static GuiKeyWord kw_193 [4]
- static GuiKeyWord kw_194 [8]
- static GuiKeyWord kw_195 [2]
- static GuiKeyWord kw_196 [4]
- static GuiKeyWord kw_197 [10]
- static GuiKeyWord kw_198 [6]
- static GuiKeyWord kw_199 [3]
- static GuiKeyWord kw_200 [10]
- static GuiKeyWord kw_201 [2]
- static GuiKeyWord kw_202 [8]
- static GuiKeyWord kw_203 [6]
- static GuiKeyWord kw_204 [6]
- static GuiKeyWord kw_205 [29]
- static GuiKeyWord kw_206 [6]
- static KeyWord kw_1 [3]
- static KeyWord kw_2 [1]
- static KeyWord kw_3 [4]
- static KeyWord kw_4 [2]
- static KeyWord kw_5 [7]
- static KeyWord kw_6 [7]
- static KeyWord kw_7 [9]
- static KeyWord kw_8 [4]
- static KeyWord kw_9 [10]
- static KeyWord kw_18 [2]
- static KeyWord kw_33 [1]
- static KeyWord kw_41 [1]
- static KeyWord kw_58 [2]
- static KeyWord kw_60 [2]
- static KeyWord kw_68 [2]
- static KeyWord kw_74 [2]
- static KeyWord kw_78 [2]
- static KeyWord kw_89 [3]
- static KeyWord kw_95 [1]
- static KeyWord kw_100 [2]
- static KeyWord kw_107 [4]
- static KeyWord kw_114 [2]
- static KeyWord kw_128 [2]
- static KeyWord kw_144 [2]
- static KeyWord kw_145 [2]
- static KeyWord kw_160 [8]
- static KeyWord kw_166 [3]
- static KeyWord kw_173 [1]
- static KeyWord kw_177 [3]
- static KeyWord kw_193 [4]
- static KeyWord kw_194 [8]
- static KeyWord kw_196 [4]
- static KeyWord kw_198 [6]
- static KeyWord kw_207 [4]
- static KeyWord kw_208 [10]
- static KeyWord kw_209 [6]
- static KeyWord kw_210 [3]
- static KeyWord kw_211 [10]
- static KeyWord kw_212 [2]
- static KeyWord kw_213 [8]
- static KeyWord kw_214 [6]
- static KeyWord kw_215 [6]
- static KeyWord kw_216 [29]
- static KeyWord kw_217 [6]
- FILE * nidrin
• static const char * aln_scaletypes [ ] = { "auto", "log", "none", 0 }
• static Var_uinfo CAUVLbl [CAUVar_Nkinds]
• static Var_uinfo DAUVLbl [DAUVar_Nkinds]
• static Var_uinfo DAURVLbl [DAURVar_Nkinds]
• static Var_uinfo CEUVLbl [CEUVar_Nkinds]
• static Var_uinfo DiscSetLbl [DiscSetVar_Nkinds]
• static VarLabelChk Vlch [ ]
• static VLstuff VLS [N_VLS]
• static int VL_aleatory [N_VLS] = { 1, 0, 1, 1 }
• static String MP_ (algebraicMappings)
• static String MP_ (idInterface)
• static String MP_ (inputFilter)
• static String MP_ (outputFilter)
• static String MP_ (parametersFile)
• static String MP_ (resultsFile)
• static String MP_ (templateDir)
• static String MP_ (workDir)
• static String2DArray MP_ (analysisComponents)
• static StringArray MP_ (analysisDrivers)
• static StringArray MP_ (templateFiles)
• static RealVectorArray MP_ (genReliabilityLevels)
• static RealVectorArray MP_ (probabilityLevels)
• static RealVectorArray MP_ (reliabilityLevels)
• static RealVectorArray MP_ (responseLevels)
• static UShortArray MP_ (expansionOrder)
• static UShortArray MP_ (quadratureOrder)
• static UShortArray MP_ (varPartitions)
• static String MP_ (centralPath)
• static String MP_ (expansionImportFile)
• static String MP_ (idMethod)
• static String MP_ (logFile)
• static String MP_ (meritFn)
• static String MP_ (modelPointer)
• static String MP_ (subMethodName)
• static String MP_ (subMethodPointer)
• static StringArray MP_ (linearEqScaleTypes)
• static StringArray MP_ (linearIneqScaleTypes)
• static StringArray MP_ (miscOptions)
• static String MP_ (approxSampleReuseFile)
• static String MP_ (idModel)
• static String MP_ (interfacePointer)
• static String MP_ (lowFidelityModelPointer)
• static String MP_ (optionalInterfRespPointer)
• static String MP_ (responsesPointer)
• static String MP_ (subMethodPointer)
• static String MP_ (truthModelPointer)
• static String MP_ (variablesPointer)
• static StringArray MP_ (primaryVarMaps)
• static StringArray MP_ (secondaryVarMaps)
• static StringArray MP_ (diagMetrics)
• static IntList MP_ (idAnalyticGrads)
• static IntList MP_ (idAnalyticHessians)
• static IntList MP_ (idNumericalGrads)
• static IntList MP_ (idNumericalHessians)
• static IntList MP_ (idQuasiHessians)
• static String MP_ (idResponses)
• static String MP_ (leastSqDataFile)
• static StringArray MP_ (primaryRespFnScaleTypes)
• static StringArray MP_ (nonlinearEqScaleTypes)
• static StringArray MP_ (nonlinearIneqScaleTypes)
• static String MP_ (responseLabels)
• static String MP_ (hybridGlobalMethodPointer)
• static String MP_ (hybridLocalMethodPointer)
• static String MP_ (methodPointer)
• static String MP_ (tabularDataFile)
• static StringArray MP_ (hybridMethodList)
• static IntArray VP_ (dsvi)
• static IntArray VP_ (ndsvi)
• static IntArray VP_ (ndsvr)
• static IntArray VP_ (nlv)
• static IntArray VP_ (nbp)
• static IntArray VP_ (npp)
• static IntArray VP_ (nssvi)
• static IntArray VP_ (nssvr)
• static IntArray VP_ (ssvi)
• static String MP_ (idVariables)
• static StringArray MP_ (continuousDesignLabels)
• static StringArray MP_ (continuousDesignScaleTypes)
• static StringArray MP_ (continuousStateLabels)
• static StringArray MP_ (discreteDesignRangeLabels)
• static StringArray MP_ (discreteDesignSetIntLabels)
• static StringArray MP_ (discreteDesignSetRealLabels)
• static StringArray MP_ (discreteStateRangeLabels)
• static StringArray MP_ (discreteStateSetIntLabels)
• static StringArray MP_ (discreteStateSetRealLabels)
• static Var_bgen var_mp_bgen []
• static Var_bgen var_mp_bgen_audr []
• static Var_bgen var_mp_bgen_audi []
• static Var_bgen var_mp_bgen_eu []
• static Var_bgen var_mp_bgen_dis []
• static VarBgen Bgen []
• static Var_bchk var_mp_bndchk []
• static Var_ibchk var_mp_ibndchk []
• ParallelLibrary dummy_lib
  dummy ParallelLibrary object used for mandatory reference initialization when a real ParallelLibrary instance is unavailable

• ProblemDescDB * Dak_pddb
  set by main(), for use in abort_handler()

• const int LARGE_SCALE = 100
  large-scale algorithm if numVars >= LARGE_SCALE

7.1.1 Detailed Description

The primary namespace for DAKOTA.

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

7.1.2 Function Documentation

7.1.2.1 CommandShell & flush (CommandShell & shell)

convenient shell manipulator function to "flush" the shell
global convenience function for manipulating the shell; invokes the class member flush function.

7.1.2.2 Real Dakota::getdist (const RealVector & x1, const RealVector & x2)

Gets the Euclidean distance between x1 and x2

7.1.2.3 Real Dakota::mindist (const RealVector & x, const RealMatrix & xset, int except)

Returns the minimum distance between the point x and the points in the set xset (compares against all points in xset except point "except"): if except is not needed, pass 0.

7.1.2.4 Real Dakota::mindistindx (const RealVector & x, const RealMatrix & xset, const IntArray & indx)

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

7.1.2.5 Real Dakota::getRmax (const RealMatrix & xset)

Gets the maximum of the min distance between each point and the rest of the set.
7.1 Dakota Namespace Reference

7.1.2.6 T Dakota::abort_handler_t (int code)

Templatized abort_handler_t method that allows for convenient return from methods that otherwise have no sensible return from error clauses. Usage: MyType& method() { return abort_handler<MyType&>(-1); }

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

sample function prototype for launching grid computing

7.1.2.8 int Dakota::stop_grid_computing ()

sample function prototype for terminating grid computing

7.1.2.9 int Dakota::perform_analysis (char * iteration_num)

sample function prototype for submitting a grid evaluation

7.1.2.10 string Dakota::asstring (const T & val)

Creates a string from the argument val using an ostringstream.

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

Parameters:

val  The value of type T to convert to a string.

Returns:

The string representation of val created using an ostringstream.

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

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

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

search function for a particular ParamResponsePair within a PRPMultiIndex
a global function to compare the interface id and variables of a particular database_pr (presumed to be in the global history list) with a passed in key of interface id and variables provided by search_pr.
7.1.2.13 PRPCacheHIter Dakota::lookup_by_val (PRPMultiIndexCache & prp_cache, const ParamResponsePair & search_pr) [inline]

ActiveSet search data within search_pr.

Lookup occurs in two steps: (1) PRPMultiIndexCache lookup based on strict equality in interface id and variables, and (2) set_compare() post-processing based on ActiveSet subset logic.

7.1.2.14 PRPQueueHIter Dakota::lookup_by_val (PRPMultiIndexQueue & prp_queue, const ParamResponsePair & search_pr) [inline]

ActiveSet search data within search_pr.

Lookup occurs in two steps: (1) PRPMultiIndexQueue lookup based on strict equality in interface id and variables, and (2) set_compare() post-processing based on ActiveSet subset logic.

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

print a restart file

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

Prints all evaluations 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_rebuild_util to_pdb dakota.rst dakota.pdb"
"dakota_rebuild_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_rebuild_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_rebuild_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.

7.1.3  Variable Documentation

7.1.3.1  const char* FIELD_NAMES[]

Initial value:

{ "numFns", "numVars", "numACV", "numADIV",
  "numADRv", "numDerivVars", "XC", "XD",
  "xDR", "xCLabels", "xDILabels",
  "xDRLabels", "directFnASV", "directFnASM",
  "directFnDVV", "directFnDVV_bool",
  "fnFlag", "gradFlag", "hessFlag",
  "fnVals", "fnGrads", "fnHessians",
  "fnLabels", "failure", "fnEvalId" }

fields to pass to Matlab in Dakota structure

7.1.3.2  const int NUMBER_OF_FIELDS = 25

number of fields in above structure

7.1.3.3  Dakota_funcs DakFuncs0

Initial value:

{ fprintf,
  abort_handler,
  dlsolver_option,
  continuous_lower_bounds1,
  continuous_upper_bounds1,
  nonlinear_ieq_constraint_lower_bounds1,
  nonlinear_ieq_constraint_upper_bounds1,
  nonlinear_eq_constraint_targets1,
  linear_ieq_constraint_lower_bounds1,
  linear_ieq_constraint_upper_bounds1,
  linear_eq_constraint_targets1,
  linear_ieq_constraint_coeffs1,
linear_eq_constraint_coeffs1,
ComputeResponses1,
GetFuncs1,
GetGrads1,
GetContVars1,
SetBestContVars1,
SetBestDiscVars1,
SetBestRespFns1,
Get_Real1,
Get_Int1,
Get_Bool1
}

7.1.3.4 GuiKeyWord kw_1[3] [static]

Initial value:

{  
   {"active_set_vector",8,0,1,0,1441},
   {"evaluation_cache",8,0,2,0,1443},
   {"restart_file",8,0,3,0,1445}
}

799 distinct keywords (plus 89 aliases)

7.1.3.5 GuiKeyWord kw_2[1] [static]

Initial value:

{  
   {"processors_per_analysis",9,0,1,0,1425,0,0.,0.,0.,0.,0,"{Number of processors per analysis} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfApplicDF"}
}

7.1.3.6 GuiKeyWord kw_3[4] [static]

Initial value:

{  
   {"abort",8,0,1,1,1431,0,0.,0.,0.,0.,0,"{CHOOSE failure mitigation}"},
   {"continuation",8,0,1,1,1437},
   {"recover",14,0,1,1,1435},
   {"retry",9,0,1,1,1433}
}

7.1.3.7 GuiKeyWord kw_4[2] [static]

Initial value:

{  
   {"copy",8,0,1,0,1419,0,0.,0.,0.,0.,0,"{Copy template files} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfApplicSC"},
   {"replace",8,0,2,0,1421,0,0.,0.,0.,0.,0,"{Replace existing files} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfApplicSC"}
}
7.1 Dakota Namespace Reference

7.1.3.8 GuiKeyWord kw_5[7]  [static]
Initial value:

```
{  
  "dir_save",0,0,3,0,1412,  
  "dir_tag",0,0,2,0,1410,  
  "directory_save",8,0,3,0,1413,0,0,,0,,0,0,"{Save work directory} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "directory_tag",8,0,2,0,1411,0,0,,0,,0,0,"{Tag work directory} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "named",11,0,1,0,1409,0,0,,0,,0,0,"{Name of work directory} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "template_directory",11,2,4,0,1415,kw_4,0,,0,,0,,0,"{Template directory} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "template_files",15,2,4,0,1417,kw_4,0,,0,,0,,0,"{Template files} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
}
```

7.1.3.9 GuiKeyWord kw_6[7]  [static]
Initial value:

```
{  
  "aprepro",8,0,4,0,1401,0,,0,,0,,0,"{Aprepro parameters file format} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "file_save",8,0,6,0,1405,0,,0,,0,,0,"{Parameters and results file saving} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "file_tag",8,0,5,0,1403,0,,0,,0,,0,"{Parameters and results file tagging} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "parameters_file",11,0,1,0,1395,0,,0,,0,,0,"{Parameters file name} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "results_file",11,0,2,0,1397,0,,0,,0,,0,"{Results file name} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "verbatim",8,0,3,0,1399,0,,0,,0,,0,"{Verbatim driver/filter invocation syntax} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
  "work_directory",8,0,7,0,1407,kw_5,0,,0,,0,,0,"{Create work directory} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfApplicSC",  
}
```

7.1.3.10 GuiKeyWord kw_7[9]  [static]
Initial value:

```
{  
  "analysis_components",15,0,1,0,1385,0,,0,,0,,0,"{Additional identifiers for use by the analysis drivers} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
  "deactivate",8,3,6,0,1439,kw_1,0,,0,,0,,0,"{Feature deactivation} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
  "direct",8,1,4,1,1423,kw_2,0,,0,,0,,0,"{CHOOSE interface type}{Direct function interface } http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
  "failure_capture",8,4,5,0,1429,kw_3,0,,0,,0,,0,"{Failure capturing} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
  "fork",8,7,4,1,1393,kw_6,0,,0,,0,,0,"{System call interface } http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
  "grid",8,0,4,1,1427,0,,0,,0,,0,"{Grid interface } http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl",  
  "input_filter",11,0,2,0,1387,0,,0,,0,,0,"{Input filter} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl",  
  "output_filter",11,0,3,0,1389,0,,0,,0,,0,"{Output filter} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
  "system",8,7,4,1,1391,kw_6,0,,0,,0,,0,"{System call interface } http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
}
```

7.1.3.11 GuiKeyWord kw_8[4]  [static]
Initial value:

```
{  
  "analysis_concurrency",9,0,3,0,1455,0,,0,,0,,0,"{Asynchronous analysis concurrency} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
  "evaluation_concurrency",9,0,1,0,1449,0,,0,,0,,0,"{Asynchronous evaluation concurrency} http://www.cs.sandia.gov/dakota/html-ref/InterfCommands.html#InterfIndControl",  
}
```
7.1.3.12 GuiKeyWord kw_9[10] [static]

Initial value:

{ "algebraic_mappings",11,0,2,0,1381,0,0,0,0,0,"{Algebraic mappings file} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfAlgebraic" },
{ "analysis_drivers",15,9,3,0,1383,kw_7,0,0,0,0,0,"{Analysis drivers} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfApplic" },
{ "analysis_self_scheduling",8,0,8,0,1465,0,0,0,0,0,"{CHOOSE analysis sched.}{Self scheduling of analysis} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" },
{ "analysis_servers",9,0,7,0,1463,0,0,0,0,0,"{Number of analysis servers} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" },
{ "analysis_static_scheduling",8,0,8,0,1467,0,0,0,0,0,"{Static scheduling of analyses} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" },
{ "asynchronous",8,4,4,0,1447,kw_8,0,0,0,0,0,"{Asynchronous interface usage} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" },
{ "evaluation_self_scheduling",8,0,6,0,1459,0,0,0,0,0,"{CHOOSE analysis sched.}{Self scheduling of evaluations} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" },
{ "evaluation_servers",9,0,5,0,1457,0,0,0,0,0,"{Number of evaluation servers} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" },
{ "evaluation_static_scheduling",8,0,6,0,1461,0,0,0,0,0,"{Static scheduling of evaluations} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" },
{ "id_interface",11,0,1,0,1379,0,0,0,0,0,"{Interface set identifier} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/InterfCommands.html#InterfIndControl" } }

7.1.3.13 static KeyWord kw_10 [static]

Initial value:

{ "merit1",8,0,1,1,239,0,0,0,0,0,"{CHOOSE merit function}" },
{ "merit1_smooth",8,0,1,1,241 },
{ "merit2",8,0,1,1,243 },
{ "merit2_smooth",8,0,1,1,245,0,0,0,0,0,"@" },
{ "merit_squared",8,0,1,1,247 },
{ "merit_max",8,0,1,1,253 },
{ "merit_max_smooth",8,0,1,1,237 } }

7.1.3.14 static KeyWord kw_11 [static]

Initial value:

{ "blocking",8,0,1,1,229,0,0,0,0,0,"{CHOOSE synchronization}" },
{ "nonblocking",8,0,1,1,231,0,0,0,0,0,"@" } }

7.1.3.15 static KeyWord kw_12 [static]

Initial value:
7.1 Dakota Namespace Reference

7.1.3.16 static KeyWord kw_13 [static]

Initial value:

```
{deltas_per_variable,5,0,2,2,944},
{step_vector,14,0,1,1,943,0,0,.,0,0,0,0},
```

7.1.3.17 static KeyWord kw_14 [static]

Initial value:

```
{initial_delta,10,0,1,1,301,0,0,.,0,0,0,0},
```

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1.3.18 static KeyWord kw_15 [static]

Initial value:

```
{
  "all_dimensions",8,0,1,1,311,
  "major_dimension",8,0,1,1,309
}
```

7.1.3.19 static KeyWord kw_16 [static]

Initial value:

```
{
  "constraint_penalty",10,0,6,0,321,0,0,0,0,0,"(Constraint penalty) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS",
  "division",8,2,1,0,307,kw_15,0,0,0,"(Box subdivision approach) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDIR",
  "global_balance_parameter",10,0,2,0,313,0,0,0,0,0,"(Global search balancing parameter) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC",
  "linear_inequality_constraint_matrix",14,0,12,0,399,0,0,0,0,0,"(Linear inequality constraint matrix) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_inequality_scale_types",15,0,14,0,403,0,0,0,0,0,"(Linear inequality scaling types) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_inequality_scales",14,0,15,0,405,0,0,0,0,0,"(Linear inequality scales) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_inequality_targets",14,0,13,0,401,0,0,0,0,0,"(Linear inequality targets) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_inequality_lower_bounds",14,0,8,0,391,0,0,0,0,0,"(Linear inequality lower bounds) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_inequality_upper_bounds",14,0,9,0,393,0,0,0,0,0,"(Linear inequality upper bounds) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "local_balance_parameter",10,0,3,0,315,0,0,0,0,0,"(Local search balancing parameter) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDIR",
  "max_boxsize_limit",10,0,4,0,317,0,0,0,0,0,"(Maximum boxsize limit) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDIR",
  "min_boxsize_limit",10,0,5,0,319,0,0,0,0,0,"(Minimum boxsize limit) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC",
  "misc_options",15,0,19,0,387,0,0,0,0,0,"(Specify miscellaneous options) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC",
  "solution_accuracy",2,0,16,0,380,
  "solution_target",10,0,16,0,381,0,0,0,0,0,"(Desired solution target) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC"
}
```

7.1.3.20 static KeyWord kw_17 [static]

Initial value:

```
{
  "blend",8,0,1,1,357,
  "two_point",8,0,1,1,355,
  "uniform",8,0,1,1,359
}
```

7.1.3.21 GuiKeyWord kw_18[2] [static]

Initial value:

```
DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
```
7.1 Dakota Namespace Reference

7.1.3.22 static KeyWord kw_19  [static]

Initial value:

```
{
    "flat_file",11,0,1,1,333,
    "simple_random",8,0,1,1,329,
    "unique_random",8,0,1,1,331
}
```

7.1.3.23 static KeyWord kw_20  [static]

Initial value:

```
{
    "mutation_range",9,0,2,0,375,0,0.,0.,0.,0.,0.,"{Mutation range} http://www.cs.sandia.gov/dakota/
    "mutation_scale",10,0,1,0,373,0,0.,0.,0.,0.,0.,"{Mutation scale} http://www.cs.sandia.gov/dakota/
}
```

7.1.3.24 static KeyWord kw_21  [static]

Initial value:

```
{
    "non_adaptive",8,0,2,0,377,0,0.,0.,0.,0.,0.,"{Non-adaptive mutation flag} http://www.cs.sandia.gov/
    "offset_cauchy",8,2,1,1,369,kw_20,
    "offset_normal",8,2,1,1,367,kw_20,
    "offset_uniform",8,2,1,1,371,kw_20,
    "replace_uniform",8,0,1,1,365
}
```

7.1.3.25 static KeyWord kw_22  [static]

Initial value:

```
{
    "chc",9,0,1,1,345,0,0.,0.,0.,0.,0.,"{CHC replacement type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA",
    "elitist",9,0,1,1,347,0,0.,0.,0.,0.,0.,"{Elitist replacement type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA",
    "new_solutions_generated",9,0,2,0,349,0,0.,0.,0.,0.,0.,"{New solutions generated} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA",
    "random",9,0,1,1,343,0,0.,0.,0.,0.,0.,"{Random replacement type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"
}
```
**7.1.3.26** static KeyWord **kw_23**  

Initial value:

```
{"constraint_penalty",10,0,9,0,379,0,0,0,0,0,0,"(Constraint penalty) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYSW"},
{"crossover_rate",10,0,5,0,351,0,0,0,0,0,"(Crossover rate) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"crossover_type",8,3,6,0,353,kw_17,0,0,0,0,"(Crossover type) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"fitness_type",8,2,3,0,335,kw_18,0,0,0,0,"(Fitness type) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"initialization_type",8,3,2,0,327,kw_19,0,0,0,0,"(Initialization type) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"linear_equality_constraint_matrix",14,0,15,0,399,0,0,0,0,0,"(Linear equality constraint matrix) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_equality_scale_types",15,0,17,0,403,0,0,0,0,0,"(Linear equality scale types) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_equality_scales",14,0,18,0,405,0,0,0,0,0,"(Linear equality scales) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_equality_targets",14,0,16,0,401,0,0,0,0,0,"(Linear equality targets) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_inequality_constraint_matrix",14,0,10,0,389,0,0,0,0,0,"(Linear inequality constraint matrix) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_inequality_lower_bounds",14,0,11,0,391,0,0,0,0,0,"(Linear inequality lower bounds) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_inequality_scale_types",15,0,13,0,395,0,0,0,0,0,"(Linear inequality scale types) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_inequality_scales",14,0,14,0,397,0,0,0,0,0,"(Linear inequality scales) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"linear_inequality_upper_bounds",14,0,12,0,393,0,0,0,0,0,"(Linear inequality upper bounds) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl"},
{"misc_options",15,0,22,0,387,0,0,0,0,0,"(Specify miscellaneous options) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC"},
{"mutation_rate",10,0,7,0,361,0,0,0,0,0,"(Mutation rate) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"mutation_type",8,5,8,0,363,kw_21,0,0,0,0,"(Mutation type) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"population_size",9,0,1,0,325,0,0,0,0,0,"(Number of population members) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"replacement_type",8,4,4,0,341,kw_22,0,0,0,0,0,"(Replacement type) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"seed",9,0,20,0,383,0,0,0,0,0,"(Random seed) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYEA"},
{"show_misc_options",8,0,21,0,385,0,0,0,0,0,"(Show miscellaneous options) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC"},
{"solution_accuracy",2,0,19,0,380,0,0,0,0,0,"(Solution accuracy) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC"},
{"solution_target",10,0,19,0,381,0,0,0,0,0,"(Solution target) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYDC"}
```

**7.1.3.27** static KeyWord **kw_24**  

Initial value:

```
{"adaptive_pattern",8,0,1,1,275},
{"basic_pattern",8,0,1,1,277},
{"multi_step",8,0,1,1,273}
```

**7.1.3.28** static KeyWord **kw_25**  

Initial value:

```
{"coordinate",8,0,1,1,263},
{"simplex",8,0,1,1,265}
```

**7.1.3.29** static KeyWord **kw_26**  

Initial value:
7.1 Dakota Namespace Reference

7.1.3.30 static KeyWord kw_27 [static]

Initial value:

```
{"blocking",8,0,1,1,281},
{"nonblocking",8,0,1,1,283}
```

7.1.3.31 static KeyWord kw_28 [static]

Initial value:

```
{"blocking",8,0,1,0,293,0,0,0,0,0},
{"nonblocking",8,0,1,0,293,0,0,0,0,0}
```

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1.3.32 static KeyWord kw_29 [static]

Initial value:

```

```

7.1.3.33 static KeyWord kw_30 [static]

Initial value:

```

```

7.1.3.34 static KeyWord kw_31 [static]

Initial value:

```

DAKOTA Namespace Documentation

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1 Dakota Namespace Reference

7.1.3.35 static KeyWord kw_32 [static]

Initial value:

```json
{
"linear_equality_constraint_matrix",14,0,7,0,399,0,0.,0.,0.,0.,"{Linear equality coefficient matrix} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_equality_scale_types",15,0,9,0,403,0,0.,0.,0.,0.,"{Linear equality scaling types} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_equality_scales",14,0,10,0,405,0,0.,0.,0.,0.,"{Linear equality scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_equality_targets",14,0,8,0,401,0,0.,0.,0.,0.,"{Linear equality targets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_inequality_constraint_matrix",14,0,2,0,389,0,0.,0.,0.,0.,"{Linear inequality coefficient matrix} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_inequality_lower_bounds",14,0,3,0,391,0,0.,0.,0.,0.,"{Linear inequality lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_inequality_scale_types",15,0,5,0,395,0,0.,0.,0.,0.,"{Linear inequality scaling types} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_inequality_scales",14,0,6,0,397,0,0.,0.,0.,0.,"{Linear inequality scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"linear_inequality_upper_bounds",14,0,4,0,393,0,0.,0.,0.,0.,"{Linear inequality upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
"optimization_type",8,2,1,0,157,kw_31,0.,0.,0.,0.,"{Optimization type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodDOTDC"
}
```

7.1.3.36 GuiKeyWord kw_33[1] [static]

Initial value:

```json
{
"seed",9,0,7,0,765,0,0.,0.,0.,0.,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS"
}
```

7.1.3.37 static KeyWord kw_34 [static]

Initial value:

```json
{
"grid",8,0,1,1,751,0,0.,0.,0.,0.,"{CHOOSE trial type}"},
"halton",8,0,1,1,753,
"random",8,0,1,1,755,0,0.,0.,0.,0.,"@"
}
```

7.1.3.38 static KeyWord kw_35 [static]

Initial value:

```json
{
"fixed_seed",8,0,4,0,747,0,0.,0.,0.,0.,"{Fixed seed flag} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDSC",
"latinize",8,0,1,0,741,0,0.,0.,0.,0.,"{Latinization of samples} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE",
"num_trials",9,0,6,0,757,0,0.,0.,0.,0.,"{Number of trials } http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE"
}
```
{"quality_metrics",8,0,2,0,743,0,0,0,0,0,0,"{Quality metrics} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"samples",9,0,8,0,763,0,0,0,0,0,0,"{Number of samples} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMC},
{"seed",9,0,7,0,765,0,0,0,0,0,0,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS},
{"trial_type",8,3,5,0,749,kw_34,0,0,0,0,0,"{Trial type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"variance_based_decomp",8,0,3,0,745,0,0,0,0,0,0,"{Variance based decomposition} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE}

7.1.3.39 static KeyWord kw_36 [static]

Initial value:

{"fixed_sequence",8,0,6,0,921,0,0,0,0,0,0,"{Fixed sequence flag} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"halton",8,0,1,1,909,0,0,0,0,0,0,"{CHOOSE sequence type}",
{"hammersley",8,0,1,1,911},
{"latinize",8,0,2,0,913,0,0,0,0,0,0,"{Latinization of samples} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"prime_base",13,0,9,0,927,0,0,0,0,0,0,"{Prime bases for sequences} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"quality_metrics",8,0,3,0,915,0,0,0,0,0,0,"{Quality metrics} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"samples",9,0,5,0,919,0,0,0,0,0,0,"{Number of samples on PCE for generating statistics} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"sequence_leap",13,0,8,0,925,0,0,0,0,0,0,"{Sequence leaping indices} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"sequence_start",13,0,7,0,923,0,0,0,0,0,0,"{Sequence starting indices} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE},
{"variance_based_decomp",8,0,4,0,917,0,0,0,0,0,0,"{Variance based decomposition} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodFSUDACE}

7.1.3.40 static KeyWord kw_37 [static]

Initial value:

{"list_of_points",14,0,1,1,939,0,0,0,0,0,0,"{List of points to evaluate} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodPSLPS}

7.1.3.41 static KeyWord kw_38 [static]

Initial value:

{"num_offspring",0x19,0,2,0,537,0,0,0,0,0,0,"{Number of offspring in random shuffle crossover} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC},
{"num_parents",0x19,0,1,0,535,0,0,0,0,0,0,"{Number of parents in random shuffle crossover} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC}

7.1.3.42 static KeyWord kw_39 [static]

Initial value:

{"crossover_rate",10,0,2,0,539,0,0,0,0,0,0,"{Crossover rate} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC},
{"multi_point_binary",9,0,1,1,527,0,0,0,0,0,0,"{Multi point binary crossover} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC}
7.1 Dakota Namespace Reference

7.1.3.43 static KeyWord kw_40 [static]
Initial value:

```
{
    "flat_file", 11, 0, 1, 1, 523,
    "simple_random", 8, 0, 1, 1, 519,
    "unique_random", 8, 0, 1, 1, 521
}
```

7.1.3.44 GuiKeyWord kw_41[1] [static]
Initial value:

```
{
    "mutation_scale", 10, 0, 1, 0, 553, 0, 0., 0., 0., 0,"[Mutation scale] http://www.cs.sandia.gov/dakota/
}
```

7.1.3.45 static KeyWord kw_42 [static]
Initial value:

```
{
    "bit_random", 8, 0, 1, 1, 543,
    "mutation_rate", 10, 0, 2, 0, 555, 0, 0., 0., 0., 0,"[Mutation rate] http://www.cs.sandia.gov/dakota/
    "offset_cauchy", 8, 1, 1, 1, 549, kw_41,
    "offset_normal", 8, 1, 1, 1, 547, kw_41,
    "offset_uniform", 8, 1, 1, 1, 551, kw_41,
    "replace_uniform", 8, 0, 1, 1, 545
}
```

7.1.3.46 static KeyWord kw_43 [static]
Initial value:

```
{
    "metric_tracker", 8, 0, 1, 1, 469, 0, 0., 0., 0., 0., 0,"[Convergence type] http://www.cs.sandia.gov/dakota/
    "num_generations", 0x29, 0, 3, 0, 473, 0, 0., 0., 0., 0., 0,"[Number generations for metric_tracker convergence] http://www.cs.sandia.gov/dakota/
    "percent_change", 10, 0, 2, 0, 471, 0, 0., 0., 0., 0., 0,"[Percent change limit for metric_tracker convergence] http://www.cs.sandia.gov/dakota/
}
```
7.1.3.47  static KeyWord kw_44  [static]

Initial value:

{  
  {"domination_count",8,0,1,1,447},
  {"layer_rank",8,0,1,1,445}
}

7.1.3.48  static KeyWord kw_45  [static]

Initial value:

{  
  {"distance",14,0,1,1,465},
  {"radial",14,0,1,1,463}
}

7.1.3.49  static KeyWord kw_46  [static]

Initial value:

{  
  {"orthogonal_distance",14,0,1,1,477,0,0.,0.,0.,0.,0.,0.,0."{Post_processor distance} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGAMOGA}
}

7.1.3.50  static KeyWord kw_47  [static]

Initial value:

{  
  {"shrinkage_fraction",10,0,1,0,459},
  {"shrinkage_percentage",2,0,1,0,458}
}

7.1.3.51  static KeyWord kw_48  [static]

Initial value:

{  
  {"below_limit",10,2,1,1,457,kw_47,0.,0.,0.,0.,0.,0,"{Below limit selection} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC",
  {"elitist",8,0,1,1,451},
  {"roulette_wheel",8,0,1,1,453},
  {"unique_roulette_wheel",8,0,1,1,455}
}
7.1 Dakota Namespace Reference

7.1.3.52 static KeyWord kw_49 [static]

Initial value:

```c
{"convergence_type",8,3,4,0,467,kw_43,0.,0.,0.,0,"{Convergence type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGASOGA"},
{"crossover_type",8,5,11,0,525,kw_39,0.,0.,0.,0,"{Crossover type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"fitness_type",8,2,1,0,443,kw_44,0.,0.,0.,0,"{Fitness type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGAMOGA"},
{"initialization_type",8,3,10,0,517,kw_40,0.,0.,0.,0,"{Initialization type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"log_file",11,0,8,0,513,0.,0.,0.,0,"{Log file} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"mutation_type",8,6,12,0,541,kw_42,0.,0.,0.,0,"{Mutation type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"niching_type",8,2,3,0,461,kw_45,0.,0.,0.,0,"{Niche pressure type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGAMOGA"},
{"population_size",9,0,7,0,511,0.,0.,0.,0,"{Number of population members} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"postprocessor_type",8,1,5,0,475,kw_46,0.,0.,0.,0,"{Post_processor type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGAMOGA"},
{"print_each_pop",8,0,9,0,515,0.,0.,0.,0,"{Population output} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"replacement_type",8,4,2,0,449,kw_48,0.,0.,0.,0,"{Replacement type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"seed",9,0,6,0,765,0.,0.,0.,0,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"partitions",13,0,1,1,949,0.,0.,0.,0,"{Partitions per variable} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodPSMPS"},
{"min_boxsize_limit",10,0,2,0,771,0.,0.,0.,0,"{Min boxsize limit} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNCSUDC"},
{"solution_accuracy",2,0,1,0,768},
{"solution_target",10,0,1,0,769,0.,0.,0.,0,"{Solution Target} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNCSUDC"},
{"volume_boxsize_limit",10,0,3,0,773}
}
```

7.1.3.53 static KeyWord kw_50 [static]

Initial value:

```c
{"partitions",13,0,1,1,949,0.,0.,0.,0,"{Partitions per variable} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodPSMPS"}
```

7.1.3.54 static KeyWord kw_51 [static]

Initial value:

```c
{"min_boxsize_limit",10,0,2,0,771,0.,0.,0.,0,"{Min boxsize limit} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNCSUDC"},
{"solution_accuracy",2,0,1,0,768},
{"solution_target",10,0,1,0,769,0.,0.,0.,0,"{Solution Target} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNCSUDC"},
{"volume_boxsize_limit",10,0,3,0,773}
```

7.1.3.55 static KeyWord kw_52 [static]

Initial value:

```c
{"absolute_conv_tol",10,0,2,0,411,0.,0.,0.,0,"{Absolute function convergence tolerance} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"covariance",9,0,8,0,423,0.,0.,0.,0,"{Covariance post-processing} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"false_conv_tol",10,0,6,0,419,0.,0.,0.,0,"{False convergence tolerance} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"function_precision",10,0,1,0,409,0.,0.,0.,0,"{Relative precision in least squares terms} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"initial_trust_radius",10,0,7,0,421,0.,0.,0.,0,"{Initial trust region radius} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"regression_diagnostics",8,0,9,0,425,0.,0.,0.,0,"{Regression diagnostics post-processing} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"singular_conv_tol",10,0,4,0,415,0.,0.,0.,0,"{Singular convergence tolerance} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"singular_radius",10,0,5,0,417,0.,0.,0.,0,"{Step limit for sctol} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"},
{"x_conv_tol",10,0,3,0,413,0.,0.,0.,0,"{Convergence tolerance for change in parameter vector} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodLSNL2SOL"}
```
7.1.3.56  static KeyWord kw_53  [static]

Initial value:

```
{
  "mt19937", 8, 0, 1, 1, 709,
  "rnum2", 8, 0, 1, 1, 711
}
```

7.1.3.57  static KeyWord kw_54  [static]

Initial value:

```
{
  "rng", 8, 2, 1, 0, 707, kw_53,
  "samples", 9, 0, 3, 0, 763, 0, 0, 0,, 0,"{Number of samples} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMC",
  "seed", 9, 0, 2, 0, 765, 0, 0, 0,, 0,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS"
}
```

7.1.3.58  static KeyWord kw_55  [static]

Initial value:

```
{
  "complementary", 8, 0, 1, 1, 677, 0, 0, 0,, 0,"{CHOOSE distribution type}"
}
```

7.1.3.59  static KeyWord kw_56  [static]

Initial value:

```
{
  "num_gen_reliability_levels", 13, 0, 1, 0, 685, 0, 0, 0,, 0,"{Number of generalized reliability levels}"
}
```

7.1.3.60  static KeyWord kw_57  [static]

Initial value:

```
{
  "num_probability_levels", 13, 0, 1, 0, 681, 0, 0, 0,, 0,"{Number of probability levels} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"
}
```
7.1 Dakota Namespace Reference

7.1.3.61 GuiKeyWord kw_58[2] [static]

Initial value:

```
{"mt19937",8,0,1,1,689},
{"rnum2",8,0,1,1,691}
```

7.1.3.62 static KeyWord kw_59 [static]

Initial value:

```
{"gen_reliabilities",8,0,1,1,671,0,0,0,0,0,0,"[CHOOSE statistic]"},
{"probabilities",8,0,1,1,669,0,0,0,0,0,0,"@"}
```

7.1.3.63 GuiKeyWord kw_60[2] [static]

Initial value:

```
{"compute",8,2,2,0,667,kw_59},
{"num_response_levels",13,0,1,0,665}
```

7.1.3.64 static KeyWord kw_61 [static]

Initial value:

```
{"distribution",8,2,5,0,673,kw_55,0,0,0,0,0,"{Distribution type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
{"ego",8,0,1,0,661},
{"gen_reliability_levels",14,1,6,0,679,kw_57,0,0,0,0,0,"{Generalized reliability levels} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
{"lhs",8,0,1,0,659},
{"probability_levels",14,1,6,0,679,kw_57,0,0,0,0,0,"{Probability levels} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
{"response_levels",14,2,2,0,663,kw_60},
{"rng",8,2,8,0,687,kw_58,0,0,0,0,0,"{Random seed generator} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDGlobalIntervalEst"},
{"samples",9,0,4,0,763,0,0,0,0,0,0,"{Number of samples} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMC"},
{"seed",9,0,3,0,765,0,0,0,0,0,0,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS"}
```

7.1.3.65 static KeyWord kw_62 [static]

Initial value:

```
{"mt19937",8,0,1,1,701},
{"rnum2",8,0,1,1,703}
```

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1.3.66 static KeyWord kw_63  [static]
Initial value:

\{
  \{"ego",8,0,1,0,697},
  \{"lhs",8,0,1,0,695},
  \{"rng",8,2,2,0,699,kw_62,0.,0.,0.,0,"{Random seed generator} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDGlobalEvid"},
  \{"samples",9,0,4,0,763,0.,0.,0.,0,"{Number of samples} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMC"},
  \{"seed",9,0,3,0,765,0.,0.,0.,0,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS"}
\}

7.1.3.67 static KeyWord kw_64  [static]
Initial value:

\{
  \{"complementary",8,0,1,1,897,0.,0.,0.,0,"[CHOOSE distribution type]"},
  \{"cumulative",8,0,1,1,895,0.,0.,0.,0,"@"}
\}

7.1.3.68 static KeyWord kw_65  [static]
Initial value:

\{
  \{"num_gen_reliability_levels",13,0,1,0,905\}
\}

7.1.3.69 static KeyWord kw_66  [static]
Initial value:

\{
  \{"num_probability_levels",13,0,1,0,901\}
\}

7.1.3.70 static KeyWord kw_67  [static]
Initial value:

\{
  \{"gen_reliabilities",8,0,1,1,891,0.,0.,0.,0,"[CHOOSE statistic]"},
  \{"probabilities",8,0,1,1,889,0.,0.,0.,0,"@"}
\}
7.1 Dakota Namespace Reference

7.1.3.71 GuiKeyWord kw_68[2] [static]

Initial value:

```
{
    "compute", 8, 2, 0, 887, kw_67,
    "num_response_levels", 13, 0, 1, 0, 885
}
```

7.1.3.72 static KeyWord kw_69 [static]

Initial value:

```
{
    "mt19937", 8, 0, 1, 1, 879,
    "rnum2", 8, 0, 1, 1, 881
}
```

7.1.3.73 static KeyWord kw_70 [static]

Initial value:

```
{
    "all_variables", 8, 0, 2, 0, 873, 0, 0, 0, 0,"{All variables flag} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDGlobalRel",
    "distribution", 8, 2, 6, 0, 893, kw_64,
    "gen_reliability_levels", 14, 1, 8, 0, 903, kw_65,
    "probability_levels", 14, 1, 7, 0, 899, kw_66,
    "response_levels", 14, 2, 5, 0, 883, kw_68,
    "rng", 8, 2, 4, 0, 877, kw_69,
    "seed", 9, 0, 3, 0, 875, 0, 0, 0, 0,"{Random seed} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC",
    "u_gaussian_process", 8, 0, 1, 1, 871, 0, 0, 0, 0,"[CHOOSE approx. type]*",
    "x_gaussian_process", 8, 0, 1, 1, 869
}
```

7.1.3.74 static KeyWord kw_71 [static]

Initial value:

```
{
    "gen_reliabilities", 8, 0, 1, 1, 655, 0, 0, 0, 0,"[CHOOSE statistic]*",
    "probabilities", 8, 0, 1, 1, 653, 0, 0, 0, 0,"0"
}
```

7.1.3.75 static KeyWord kw_72 [static]

Initial value:

```
{
    "compute", 8, 2, 2, 0, 651, kw_71,
    "num_response_levels", 13, 0, 1, 0, 649
}
```
7.1.3.76 static KeyWord kw_73 [static]

Initial value:

{
  "distribution", 8, 2, 4, 0, 673, kw_55, 0., 0., 0., 0., "[Distribution type] http://www.cs.sandia.gov/dakota/
  "gen_reliability_levels", 14, 1, 6, 0, 683, kw_56, 0., 0., 0., 0., "[Generalized reliability levels] http://www.cs.sandia.gov/
  "probability_levels", 14, 1, 5, 0, 679, kw_57, 0., 0., 0., 0., "[Probability levels] http://www.cs.sandia.gov/
  "response_levels", 14, 2, 1, 0, 647, kw_72,
  "rng", 8, 2, 7, 0, 687, kw_58, 0., 0., 0., 0., "[Random seed generator] http://www.cs.sandia.gov/dakota/
  "samples", 9, 0, 3, 0, 763, 0., 0., 0., 0., "[Number of samples] http://www.cs.sandia.gov/dakota/
  "seed", 9, 0, 2, 0, 765, 0., 0., 0., 0., "[Random seed for stochastic pattern search] http://www.cs.sandia.gov/
}

7.1.3.77 GuiKeyWord kw_74[2] [static]

Initial value:

{
  "complementary", 8, 0, 1, 1, 803, 0., 0., 0., 0., 0., 0, "[CHOOSE distribution type]"},
  "cumulative", 8, 0, 1, 1, 801, 0., 0., 0., 0., 0, "@"
}

7.1.3.78 static KeyWord kw_75 [static]

Initial value:

{
  "num_gen_reliability_levels", 13, 0, 1, 0, 797
}

7.1.3.79 static KeyWord kw_76 [static]

Initial value:

{
  "num_probability_levels", 13, 0, 1, 0, 793
}

7.1.3.80 static KeyWord kw_77 [static]

Initial value:

{
  "gen_reliabilities", 8, 0, 1, 1, 789, 0., 0., 0., 0., 0., 0, "[CHOOSE statistic]"},
  "probabilities", 8, 0, 1, 1, 787, 0., 0., 0., 0., 0, "@"
}
7.1.3.81 GuiKeyWord kw_78[2] [static]
Initial value:

```
{ "compute", 8, 2, 2, 0, 785, kw_77 },
{ "num_response_levels", 13, 0, 1, 0, 783 }
```

7.1.3.82 static KeyWord kw_79 [static]
Initial value:

```
{ "distribution", 8, 2, 5, 0, 799, kw_74 },
{ "gen_reliability_levels", 14, 1, 4, 0, 795, kw_75 },
{ "nip", 8, 0, 1, 0, 779 },
{ "probability_levels", 14, 1, 3, 0, 791, kw_76 },
{ "response_levels", 14, 2, 2, 0, 781, kw_78 },
{ "sqp", 8, 0, 1, 0, 777 }
```

7.1.3.83 static KeyWord kw_80 [static]
Initial value:

```
{ "nip", 8, 0, 1, 0, 809 },
{ "sqp", 8, 0, 1, 0, 807 }
```

7.1.3.84 static KeyWord kw_81 [static]
Initial value:

```
{ "adapt_import", 8, 0, 1, 1, 843, 0, 0., 0., 0., 0., 0., "[CHOOSE refinement type]" },
{ "import", 8, 0, 1, 1, 841 },
{ "mm_adapt_import", 8, 0, 1, 1, 845 },
{ "samples", 9, 0, 2, 0, 847, 0, 0., 0., 0., 0., 0., "[Refinement samples] http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDLocalRel" },
{ "seed", 9, 0, 3, 0, 849, 0, 0., 0., 0., 0., 0., "[Random seed for stochastic pattern search] http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYSW" }
```

7.1.3.85 static KeyWord kw_82 [static]
Initial value:

```
```

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1.3.86 static KeyWord kw_83 [static]
Initial value:
{
  "nip", 8, 0, 2, 0, 831},
  "no_approx", 8, 0, 1, 1, 827, 0, 0, 0, 0, "{CHOOSE MPP search method}"),
  "sgp", 8, 0, 2, 0, 829},
  "u_taylor_mean", 8, 0, 1, 1, 817),
  "u_taylor_mpp", 8, 0, 1, 1, 821},
  "u_two_point", 8, 0, 1, 1, 825},
  "x_taylor_mean", 8, 0, 1, 1, 815),
  "x_taylor_mpp", 8, 0, 1, 1, 819),
  "x_two_point", 8, 0, 1, 1, 823
}

7.1.3.87 static KeyWord kw_84 [static]
Initial value:
{
  "num_reliability_levels", 13, 0, 1, 0, 865}
}

7.1.3.88 static KeyWord kw_85 [static]
Initial value:
{
  "gen_reliabilities", 8, 0, 1, 1, 861, 0, 0, 0, 0, "{CHOOSE statistic}"),
  "probabilities", 8, 0, 1, 1, 857, 0, 0, 0, 0, "{statistic}"),
  "reliabilities", 8, 0, 1, 1, 859)
}

7.1.3.89 static KeyWord kw_86 [static]
Initial value:
{
  "compute", 8, 3, 2, 0, 855, kw_85},
  "num_response_levels", 13, 0, 1, 0, 853
}
7.1 Dakota Namespace Reference

7.1.3.90  static KeyWord kw_87  [static]

Initial value:

```
{ "distribution", 8, 2, 5, 0, 893, kw_64 },
{ "gen_reliability_levels", 14, 1, 7, 0, 903, kw_65 },
{ "integration", 8, 3, 2, 0, 833, kw_82, 0., 0., 0., 0., "\{Integration method\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDLocalRel" },
{ "mpp_search", 8, 9, 1, 0, 813, kw_83, 0., 0., 0., 0., "\{MPP search type\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDLocalRel" },
{ "probability_levels", 14, 1, 6, 0, 899, kw_66 },
{ "reliability_levels", 14, 1, 4, 0, 863, kw_84 },
{ "response_levels", 14, 2, 3, 0, 851, kw_86 }
```

7.1.3.91  static KeyWord kw_88  [static]

Initial value:

```
{ "num_reliability_levels", 13, 0, 1, 0, 631, 0., 0., 0., 0., 0., 0., "\{Number of reliability levels\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD" }
```

7.1.3.92  GuiKeyWord kw_89[3]  [static]

Initial value:

```
{ "gen_reliabilities", 8, 0, 1, 1, 643, 0., 0., 0., 0., 0., 0., "\{CHOOSE statistic\}" },
{ "probabilities", 8, 0, 1, 1, 639, 0., 0., 0., 0., 0., 0., "\{\}" },
{ "reliabilities", 8, 0, 1, 1, 641 } 
```

7.1.3.93  static KeyWord kw_90  [static]

Initial value:

```
{ "compute", 8, 3, 2, 0, 637, kw_89, 0., 0., 0., 0., 0., 0., "\{Target statistics for response levels\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD" },
{ "num_response_levels", 13, 0, 1, 0, 635, 0., 0., 0., 0., 0., 0., "\{Number of response levels\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD" }
```

7.1.3.94  static KeyWord kw_91  [static]

Initial value:

```
{ "expansion_order", 13, 0, 2, 1, 581, 0., 0., 0., 0., 0., 0., "\{Expansion order\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE" },
{ "expansion_terms", 9, 0, 2, 1, 583, 0., 0., 0., 0., 0., 0., "\{Expansion terms\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE" },
{ "reuse_samples", 8, 0, 1, 0, 573, 0., 0., 0., 0., 0., 0., "\{Reuse samples flag for PCE coefficient estimation\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE" }
```
7.1.3.95 static KeyWord kw_92 [static]

Initial value:

```
{
    "expansion_order",13,0,2,1,581,0,0.,0.,0.,0.,0,"{Expansion order} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
    "expansion_terms",9,0,2,1,583,0,0.,0.,0.,0.,0,"{Expansion terms} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
}
```

7.1.3.96 static KeyWord kw_93 [static]

Initial value:

```
{
    "expansion_order",13,0,2,1,581,0,0.,0.,0.,0,"{Expansion order} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
    "expansion_terms",9,0,2,1,583,0,0.,0.,0.,0.,0,"{Expansion terms} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
    "incremental_lhs",8,0,1,0,577,0,0.,0.,0.,0.,0,"{Incremental LHS flag for PCE coefficient estimation} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
}
```

7.1.3.97 static KeyWord kw_94 [static]

Initial value:

```
{
    "lhs",8,0,1,1,587,0,0.,0.,0.,0,"{CHOOSE sample type}"
    "random",8,0,1,1,588
}
```

7.1.3.98 GuiKeyWord kw_95[1] [static]

Initial value:

```
{
    "dimension_preference",14,0,1,0,567,0,0.,0.,0.,0.,0,"{Sparse grid dimension preference} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD",
}
```

7.1.3.99 static KeyWord kw_96 [static]

Initial value:

```
{
    "all_variables",8,0,12,0,625,0,0.,0.,0.,0,"{All variables flag} http://www.cs.sandia.gov/dakota/",
    "askey",8,0,1,0,559,
    "collocation_points",9,3,2,1,569,kw_91,0,0.,0.,0.,0,"{CHOOSE PC control type}{Number of collocation points} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
    "collocation_ratio",10,3,2,1,571,kw_91,0,0.,0.,0.,0."{Collocation point oversampling ratio for PCE} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
    "distribution",8,2,6,0,675,kw_55,0,0.,0.,0.,0,"{Distribution type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE",
    "expansion_import_file",11,2,2,1,579,kw_92,0,0.,0.,0.,0,"{File name for import of PCE coefficients} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDPCE"
```
7.1 Dakota Namespace Reference

7.1.3.100 static KeyWord kw_97 [static]

Initial value:

```
{
    "previous_samples", 9, 0, 1, 1, 621, 0, 0, 0, 0, 0
}
```

7.1.3.101 static KeyWord kw_98 [static]

Initial value:

```
{
    "incremental_lhs", 8, 1, 1, 1, 617, kw_97, 0, 0, 0, 0,
    "incremental_random", 8, 1, 1, 1, 619, kw_97,
    "lhs", 8, 0, 1, 1, 615, 0, 0, 0, 0, 0,
    "random", 8, 0, 1, 1, 613
}
```

7.1.3.102 static KeyWord kw_99 [static]

Initial value:

```
{
    "all_variables", 8, 0, 11, 0, 625, 0, 0, 0, 0, 0,
    "distribution", 8, 2, 5, 0, 673, kw_55, 0, 0, 0, 0,
    "fixed_seed", 8, 0, 12, 0, 627, 0, 0, 0, 0, 0,
    "gen_reliability_levels", 14, 1, 7, 0, 679, kw_57, 0, 0, 0, 0,
    "probability_levels", 14, 1, 7, 0, 679, kw_57, 0, 0, 0, 0,
    "reliability_levels", 14, 1, 9, 0, 629, kw_98, 0, 0, 0, 0,
    "response_levels", 14, 2, 10, 0, 633, kw_90, 0, 0, 0, 0,
    "rng", 8, 2, 8, 0, 678, kw_58, 0, 0, 0, 0,
    "sample_type", 8, 4, 1, 0, 611, kw_98, 0, 0, 0, 0,
    "samples", 9, 0, 4, 0, 763, 0, 0, 0, 0, 0,
    "seed", 9, 0, 3, 0, 765, 0, 0, 0, 0, 0,
    "variance_based_decomp", 8, 0, 2, 0, 623, 0, 0, 0, 0, 0
}
```

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1.3.103 GuiKeyWord kw_100[2]  [static]

Initial value:

```c
{
  {"lhs",8,0,1,1,605,0,0.,0.,0.,0,"@[CHOOSE sample type]"},
  {"random",8,0,1,1,607}
}
```

7.1.3.104 static KeyWord kw_101  [static]

Initial value:

```c
{
  {"dimension_preference",14,0,1,0,601,0,0.,0.,0.,0,"{Sparse grid dimension preference} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDSC"},
}
```

7.1.3.105 static KeyWord kw_102  [static]

Initial value:

```c
{
  {"all_variables",8,0,12,0,625,0.,0.,0.,0.,0.,0,"{All variables flag} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMC"},
  {"askey",8,0,1,0,593},
  {"distribution",8,2,6,0,673,kw_55,0.,0.,0.,0,"{Distribution type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
  {"fixed_seed",8,0,13,0,627,0.,0.,0.,0.,0,"{Fixed seed flag} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMC"},
  {"gen_reliability_levels",14,1,8,0,683,kw_56,0.,0.,0.,0,"{Generalized reliability levels} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
  {"probability_levels",14,1,7,0,679,kw_57,0.,0.,0.,0.,0,"{Probability levels} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
  {"reliability_levels",14,1,10,0,629,kw_88,0.,0.,0.,0.,0.,0,"{Reliability levels} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
  {"quadrature_order",13,0,2,1,597,0.,0.,0.,0.,0,"{CHOOSE accuracy control} {Quadrature order for collocation points} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDSC"},
  {"response_levels",14,2,11,0,633,kw_90,0.,0.,0.,0.,0.,0,"{Response levels} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
  {"rng",8,2,9,0,687,kw_58,0.,0.,0.,0.,0.,0,"{Random seed generator} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMGlobalIntervalEst"},
  {"sample_type",8,2,3,0,603,kw_100,0.,0.,0.,0.,0.,0,"{Sampling type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonD"},
  {"samples",9,0,5,0,763,0.,0.,0.,0.,0.,0,"{Number of samples} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNonDMC"},
  {"seed",9,0,4,0,765,0.,0.,0.,0.,0.,0,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS"},
  {"wiener",8,0,1,0,595}
}
```

7.1.3.106 static KeyWord kw_103  [static]

Initial value:

```c
{
  {"misc_options",15,0,1,0,429}
}
```
Initial value:

```json
{  
  "function_precision",10,0,2,0,175,0,0,0,0,0,"{Function precision} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC",
  "linear_equality_constraint_matrix",14,0,9,0,399,0,0,0,0,0,"{Linear equality coefficient matrix} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scale_types",15,0,11,0,403,0,0,0,0,0,"{Linear equality scaling types} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scales",14,0,12,0,405,0,0,0,0,0,"{Linear equality scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_targets",14,0,10,0,401,0,0,0,0,0,"{Linear equality targets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_lower_bounds",14,0,5,0,391,0,0,0,0,0,"{Linear inequality lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_upper_bounds",14,0,6,0,393,0,0,0,0,0,"{Linear inequality upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_constraint_matrix",14,0,4,0,389,0,0,0,0,0,"{Linear inequality constraint matrix} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scale_types",15,0,4,0,395,0,0,0,0,0,"{Linear inequality scaling types} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scales",14,0,8,0,397,0,0,0,0,0,"{Linear inequality scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_targets",14,0,7,0,401,0,0,0,0,0,"{Linear equality targets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_lower_bounds",14,0,2,0,391,0,0,0,0,0,"{Linear inequality lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_upper_bounds",14,0,6,0,393,0,0,0,0,0,"{Linear inequality upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_constraint_matrix",14,0,0,0,389,0,0,0,0,0,"{Linear inequality constraint matrix} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scale_types",15,0,0,0,395,0,0,0,0,0,"{Linear inequality scaling types} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scales",14,0,5,0,397,0,0,0,0,0,"{Linear inequality scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_targets",14,0,4,0,401,0,0,0,0,0,"{Linear equality targets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_lower_bounds",14,0,3,0,391,0,0,0,0,0,"{Linear inequality lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_upper_bounds",14,0,2,0,393,0,0,0,0,0,"{Linear inequality upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "verify_level",9,0,1,0,173,0,0,0,0,0,"{Gradient verification level} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNPSOLDC",
  "max_step",10,0,10,0,209,0,0,0,0,0,"{Maximum step size} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC",
  "verify_level",9,0,1,0,173,0,0,0,0,0,"{Gradient verification level} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodNPSOLDC",
  "linear_equality_upper_bounds",14,0,6,0,393,0,0,0,0,0,"{Linear inequality upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scales",14,0,8,0,397,0,0,0,0,0,"{Linear inequality scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_scale_types",15,0,4,0,395,0,0,0,0,0,"{Linear inequality scaling types} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_targets",14,0,7,0,401,0,0,0,0,0,"{Linear equality targets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_lower_bounds",14,0,2,0,391,0,0,0,0,0,"{Linear inequality lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "linear_equality_upper_bounds",14,0,6,0,393,0,0,0,0,0,"{Linear inequality upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
  "max_step",10,0,10,0,209,0,0,0,0,0,"{Maximum step size} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC"
}
```
7.1.3.110  GuiKeyWord kw_107[4]  [static]

Initial value:

```
{
    "gradient_based_line_search", 8, 0, 1, 1, 195, 0, 0, 0, 0, 0, 0, "[CHOOSE line search type]",
    "tr_pds", 8, 0, 1, 1, 199,
    "trust_region", 8, 0, 1, 1, 197,
    "value_based_line_search", 8, 0, 1, 1, 193
}
```

7.1.3.111  static KeyWord kw_108  [static]

Initial value:

```
{
    "centering_parameter", 10, 0, 5, 0, 207, 0, 0, 0, 0, 0, 0, "[Centering parameter] http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC",
    "central_path", 11, 0, 3, 0, 203, 0, 0, 0, 0, 0, 0, "[Central path] http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC",
    "gradient_tolerance", 10, 0, 16, 0, 211, 0, 0, 0, 0, 0, 0, "[Gradient tolerance] http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC",
    "linear_equality_coefficient_matrix", 14, 0, 11, 0, 399, 0, 0, 0, 0, 0, 0, "(Linear equality coefficient matrix) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "linear_equality_scale_types", 15, 0, 13, 0, 403, 0, 0, 0, 0, 0, 0, "(Linear equality scaling types) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "linear_equality_scales", 14, 0, 14, 0, 405, 0, 0, 0, 0, 0, 0, "(Linear equality scales) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "linear_equality_targets", 14, 0, 12, 0, 401, 0, 0, 0, 0, 0, 0, "(Linear equality targets) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "linear_inequality_coefficient_matrix", 14, 0, 6, 0, 389, 0, 0, 0, 0, 0, 0, "(Linear inequality coefficient matrix) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "linear_inequality_scale_types", 15, 0, 9, 0, 395, 0, 0, 0, 0, 0, 0, "(Linear inequality scaling types) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "linear_inequality_scales", 14, 0, 10, 0, 397, 0, 0, 0, 0, 0, 0, "(Linear inequality scales) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "linear_inequality_upper_bounds", 14, 0, 8, 0, 393, 0, 0, 0, 0, 0, 0, "(Linear inequality upper bounds) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodIndControl",
    "max_step", 10, 0, 15, 0, 209, 0, 0, 0, 0, 0, 0, "(Maximum step size) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC",
    "merit_function", 11, 0, 2, 0, 201, 0, 0, 0, 0, 0, 0, "(Merit function) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodAPPSDC",
    "search_method", 8, 4, 1, 0, 191, 0, 0, 0, 0, 0, 0, "(Search method) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC",
    "steplength_to_boundary", 10, 0, 4, 0, 205, 0, 0, 0, 0, 0, 0, "(Steplength to boundary) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodOPTPPDC"
}
```

7.1.3.112  static KeyWord kw_109  [static]

Initial value:

```
{
    "debug", 8, 0, 1, 1, 67, 0, 0, 0, 0, 0, 0, "[CHOOSE output level]",
    "quiet", 8, 0, 1, 1, 71,
    "silent", 8, 0, 1, 1, 73,
    "verbose", 8, 0, 1, 1, 69
}
```

7.1.3.113  static KeyWord kw_110  [static]

Initial value:

```
{
    "partitions", 13, 0, 1, 0, 761, 0, 0, 0, 0, 0, 0, "(Number of partitions) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodPSUADE"
}
```
7.1 Dakota Namespace Reference

7.1.3.114 static KeyWord kw_111 [static]

Initial value:

```
{
  "num_generations", 0x29, 0, 2, 0, 509,
  "percent_change", 10, 0, 1, 0, 507
}
```

7.1.3.115 static KeyWord kw_112 [static]

Initial value:

```
{
  "num_generations", 0x29, 0, 2, 0, 503, 0, 0., 0., 0., 0,"{Number of generations (for convergence test) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGASOGA",
  "percent_change", 10, 0, 1, 0, 501, 0, 0., 0., 0., 0,"{Percent change in fitness} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGASOGA"
}
```

7.1.3.116 static KeyWord kw_113 [static]

Initial value:

```
{
  "average_fitness_tracker", 8, 2, 1, 1, 505, kw_111,
  "best_fitness_tracker", 8, 2, 1, 1, 499, kw_112
}
```

7.1.3.117 GuiKeyWord kw_114[2] [static]

Initial value:

```
{
  "constraint_penalty", 10, 0, 2, 0, 485, 0, 0., 0., 0., 0,"{Constraint penalty in merit function} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGASOGA",
  "merit_function", 8, 0, 1, 1, 483
}
```

7.1.3.118 static KeyWord kw_115 [static]

Initial value:


7.1.3.119 static KeyWord kw_116

Initial value:

```
{{"convergence_type",8,2,3,0,497,kw_113},
{"crossover_type",8,5,9,0,525,kw_39,0.,0.,0.,0,"{Crossover type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"fitness_type",8,2,1,0,481,kw_114,0.,0.,0.,0,"{Fitness type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGASOGA"},
{"initialization_type",8,3,8,0,517,kw_40,0.,0.,0.,0,"{Initialization type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"log_file",11,0,6,0,513,0.,0.,0.,0,"{Log file} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"mutation_type",8,6,10,0,541,kw_42,0.,0.,0.,0,"{Mutation type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"population_size",9,0,4,0,765,0.,0.,0.,0.,0,"{Number of population members} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"print_each_pop",8,0,7,0,515,0.,0.,0.,0.,0,"{Population output} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"replacement_type",8,4,2,0,487,kw_115,0.,0.,0.,0.,0,"{Replacement type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodJEGADC"},
{"seed",9,0,4,0,765,0.,0.,0.,0.,0,"{Random seed for stochastic pattern search} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodCOLINYPS"}
```

7.1.3.120 static KeyWord kw_117

Initial value:

```
{{"approx_method_name",11,0,1,1,433,0.,0.,0.,0.,0,"{CHOOSE sub-method ref.}{Approximate sub-problem minimization method name} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodSBG"},
{"approx_method_pointer",11,0,1,1,435,0.,0.,0.,0.,0,"{CHOOSE sub-method ref.}{Approximate sub-problem minimization method pointer} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodSBG"},
{"replace_points",8,0,2,0,437,0.,0.,0.,0.,0,"{Replace points used in surrogate construction with best points} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/MethodCommands.html#MethodSBG"}
```

7.1.3.121 static KeyWord kw_118

Initial value:

```
{{"filter",8,0,1,1,141,0.,0.,0.,0.,0,"{CHOOSE acceptance logic}"},
{"tr_ratio",8,0,1,1,139}
```

7.1.3.122 static KeyWord kw_119

Initial value:
7.1 Dakota Namespace Reference

{ 
  {"augmented_lagrangian_objective",8,0,1,1,17,0,0,0,0,0,0,"[CHOOSE objective formulation]"}, 
  {"lagrangian_objective",8,0,1,1,119}, 
  {"linearized_constraints",8,0,2,2,123,0,0,0,0,0,"[CHOOSE constraint formulation]"}, 
  {"no_constraints",8,0,2,2,125}, 
  {"original_constraints",8,0,2,2,121,0,0,0,0,0,"@"}, 
  {"original_primary",8,0,1,1,113,0,0,0,0,0,"@"}, 
  {"single_objective",8,0,1,1,115} 
}

7.1.3.123 static KeyWord kw_120 [static]

Initial value:

{ 
  {"homotopy",8,0,1,1,145} 
}

7.1.3.124 static KeyWord kw_121 [static]

Initial value:

{ 
  {"adaptive_penalty_merit",8,0,1,1,131,0,0,0,0,0,"[CHOOSE merit function]"}, 
  {"augmented_lagrangian_merit",8,0,1,1,135,0,0,0,0,0,"@"}, 
  {"lagrangian_merit",8,0,1,1,133}, 
  {"penalty_merit",8,0,1,1,129} 
}

7.1.3.125 static KeyWord kw_122 [static]

Initial value:

{ 
  {"contract_threshold",10,0,3,0,103,0,0,0,0,0,"[Shrink trust region if trust region ratio is less than this value]"}, 
  {"contraction_factor",10,0,5,0,107,0,0,0,0,0,"[Pattern contraction factor]"}, 
  {"expand_threshold",10,0,4,0,105,0,0,0,0,0,"[Expand trust region if trust region ratio is greater than this value]"}, 
  {"expansion_factor",10,0,6,0,109,0,0,0,0,0,"[Trust region expansion factor]"}, 
  {"initial_size",10,0,1,0,99,0,0,0,0,0,"[Trust region initial size (relative to bounds)]"}, 
  {"minimum_size",10,0,2,0,101,0,0,0,0,0,"[Trust region minimum size]"} 
}

7.1.3.126 static KeyWord kw_123 [static]

Initial value:

{ 
  {"acceptance_logic",8,2,7,0,137,kw_118,0,0,0,0,"[SBL iterate acceptance logic]"} 
}
126

DAKOTA Namespace Documentation

{"approx_method_name",11,0,1,1,89,0,0.,0.,0.,0,"[CHOOSE sub-method ref.]{Approximate sub-problem
{"approx_method_pointer",11,0,1,1,91,0,0.,0.,0.,0,"{Approximate sub-problem minimization method p
{"approx_subproblem",8,7,5,0,111,kw_119,0.,0.,0.,0,"{Approximate subproblem formulation} http://w
{"constraint_relax",8,1,8,0,143,kw_120,0.,0.,0.,0,"{SBL constraint relaxation method for infeasib
{"linear_equality_constraint_matrix",14,0,14,0,399,0,0.,0.,0.,0,"{Linear equality coefficient mat
{"linear_equality_scale_types",15,0,16,0,403,0,0.,0.,0.,0,"{Linear equality scaling types} http:/
{"linear_equality_scales",14,0,17,0,405,0,0.,0.,0.,0,"{Linear equality scales} http://www.cs.sand
{"linear_equality_targets",14,0,15,0,401,0,0.,0.,0.,0,"{Linear equality targets} http://www.cs.sa
{"linear_inequality_constraint_matrix",14,0,9,0,389,0,0.,0.,0.,0,"{Linear inequality coefficient
{"linear_inequality_lower_bounds",14,0,10,0,391,0,0.,0.,0.,0,"{Linear inequality lower bounds} ht
{"linear_inequality_scale_types",15,0,12,0,395,0,0.,0.,0.,0,"{Linear inequality scaling types} ht
{"linear_inequality_scales",14,0,13,0,397,0,0.,0.,0.,0,"{Linear inequality scales} http://www.cs.
{"linear_inequality_upper_bounds",14,0,11,0,393,0,0.,0.,0.,0,"{Linear inequality upper bounds} ht
{"merit_function",8,4,6,0,127,kw_121,0.,0.,0.,0,"{Merit function} http://www.cs.sandia.gov/dakota
{"soft_convergence_limit",9,0,2,0,93,0,0.,0.,0.,0,"{Soft convergence limit for SBL iterations} ht
{"trust_region",8,6,4,0,97,kw_122,0.,0.,0.,0,"{Trust region group specification} http://www.cs.sa
{"truth_surrogate_bypass",8,0,3,0,95,0,0.,0.,0.,0,"{Flag for bypassing lower level surrogates in
}

7.1.3.127

static KeyWord kw_124 [static]

Initial value:
{

{"final_point",14,0,1,1,931,0,0.,0.,0.,0,"[CHOOSE final pt or increment]{Termination point of vec
{"num_steps",9,0,2,2,935,0,0.,0.,0.,0,"{Number of steps along vector} http://www.cs.sandia.gov/da
{"step_vector",14,0,1,1,933,0,0.,0.,0.,0,"{Step vector} http://www.cs.sandia.gov/dakota/licensing
}

7.1.3.128

static KeyWord kw_126 [static]

Initial value:
{

{"optional_interface_responses_pointer",11,0,1,0,1099,0,0.,0.,0.,0,"{Responses pointer for nested
}

7.1.3.129

static KeyWord kw_127 [static]

Initial value:
{
{"primary_response_mapping",14,0,3,0,1107,0,0.,0.,0.,0,"{Primary response mappings for
{"primary_variable_mapping",15,0,1,0,1103,0,0.,0.,0.,0,"{Primary variable mappings for
{"secondary_response_mapping",14,0,4,0,1109,0,0.,0.,0.,0,"{Secondary response mappings
{"secondary_variable_mapping",15,0,2,0,1105,0,0.,0.,0.,0,"{Secondary variable mappings
}

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010

nested mod
nested mod
for nested
for nested


7.1 Dakota Namespace Reference

7.1.3.130  GuiKeyWord kw_128[2]  [static]

Initial value:

```
{  "optional_interface_pointer",11,1,1,0,1097,kw_126,0.,0.,0.,0,"{Optional interface set pointer} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelNested"},
  "sub_method_pointer",11,4,2,1,1101,kw_127,0.,0.,0.,0,"{Sub-method pointer for nested models} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelNested"}
```

7.1.3.131  static KeyWord kw_129  [static]

Initial value:

```
{  "interface_pointer",11,0,1,0,961,0,0.,0.,0.,0,"{Interface set pointer} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSingle"}
```

7.1.3.132  static KeyWord kw_130  [static]

Initial value:

```
{  "additive",8,0,2,2,1057,0,0.,0.,0.,0,"[CHOOSE correction type]"},
  "combined",8,0,2,2,1061},
  "first_order",8,0,1,1,1053,0,0.,0.,0,"[CHOOSE correction order]"},
  "multiplicative",8,0,2,2,1059},
  "second_order",8,0,1,1,1055},
  "zeroth_order",8,0,1,1,1051}
```

7.1.3.133  static KeyWord kw_131  [static]

Initial value:

```
{  "constant",8,0,1,1,975,0,0.,0.,0.,0,"[CHOOSE trend type]"},
  "linear",8,0,1,1,977},
  "quadratic",8,0,1,1,979,0,0.,0.,0.,0,"@"}
```

7.1.3.134  static KeyWord kw_132  [static]

Initial value:

```
{  "point_selection",8,0,1,0,971,0,0.,0.,0.,0,"{GP point selection} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG"},
  "trend",8,3,2,0,973,kw_131,0.,0.,0.,0,"{GP trend function} http://www.cs.sandia.gov/dakota/1}
```
7.1.3.135  static KeyWord kw_133  [static]

Initial value:

```
{"conmin_seed", 14, 0, 2, 0, 1027, 0, 0., 0., 0., 0., 0., \"\{Kriging inital correlations\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"},
{"correlations", 14, 0, 1, 0, 1025, 0, 0., 0., 0., 0., 0., \"\{Kriging correlations\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"},
{"max_correlations", 14, 0, 4, 0, 1031, 0, 0., 0., 0., 0., 0., \"\{Kriging maximum correlations\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"},
{"max_trials", 9, 0, 3, 0, 1029, 0, 0., 0., 0., 0., 0., \"\{Kriging maximum trials\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"},
{"min_correlations", 14, 0, 5, 0, 1033, 0, 0., 0., 0., 0., 0., \"\{Kriging minimum correlations\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"}
```

7.1.3.136  static KeyWord kw_134  [static]

Initial value:

```
{"cubic", 8, 0, 1, 1, 989},
{"linear", 8, 0, 1, 987}
```

7.1.3.137  static KeyWord kw_135  [static]

Initial value:

```
{"interpolation", 8, 2, 2, 0, 985, kw_134, 0, 0., 0., 0., 0., \"\{MARS interpolation\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"},
{"max_bases", 9, 0, 1, 0, 983, 0, 0., 0., 0., 0., \"\{MARS maximum bases\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"}
```

7.1.3.138  static KeyWord kw_136  [static]

Initial value:

```
{"poly_order", 9, 0, 1, 0, 993, 0, 0., 0., 0., 0., \"\{MLS polynomial order\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"},
{"weight_function", 9, 0, 2, 0, 995, 0, 0., 0., 0., 0., \"\{MLS weight function\} http://www.cs.sandia.gov/dakota/html-ref/ModelCommands.html#ModelSurrG\"}
```

7.1.3.139  static KeyWord kw_137  [static]

Initial value:

```
{"nodes", 9, 0, 1, 0, 999, 0, 0., 0., 0., 0., \"\{ANN number nodes\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG\"},
{"random_weight", 9, 0, 3, 0, 1003, 0, 0., 0., 0., 0., \"\{ANN random weight\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG\"},
{"range", 10, 0, 2, 0, 1001, 0, 0., 0., 0., 0., \"\{ANN range\} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG\"}
```
7.1.3.140  static KeyWord kw_138  [static]

Initial value:

{
    "cubic",8,0,1,1,1021,0,0.,0,0.,"[CHOOSE polynomial order]",
    "linear",8,0,1,1,1017,
    "quadratic",8,0,1,1,1019
}

7.1.3.141  static KeyWord kw_139  [static]

Initial value:

{
    "bases",9,0,1,0,1007,0,0.,0,0.,0,"{RBF number of bases} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "max_pts",9,0,2,0,1009,0,0.,0,0.,0,"{RBF maximum points} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "max_subsets",9,0,4,0,1013,
    "min_partition",9,0,3,0,1011,0,0.,0,0.,0,"{RBF minimum partitions} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG"
}

7.1.3.142  static KeyWord kw_140  [static]

Initial value:

{
    "all",8,0,1,1,1039,0,0.,0,0.,0,"[CHOOSE reuse scope]",
    "none",8,0,1,1,1043,
    "region",8,0,1,1,1041
}

7.1.3.143  static KeyWord kw_141  [static]

Initial value:

{
    "correction",8,6,6,0,1049,kw_130,0,0.,0,0.,0,"{Surrogate correction approach} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "dace_method_pointer",11,0,2,0,1035,0,0.,0,0.,0,"{Design of experiments method pointer} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "diagnostics",15,0,7,0,1063,0,0.,0,0.,0,"{Print diagnostic metrics about the surrogate goodness} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "gaussian_process",8,2,1,1,1023,kw_135,0,0.,0,0,0,"{Gaussian process} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "kriging",8,5,1,1,1023,kw_135,0,0.,0,0,0,"{Kriging interpolation} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "mars",8,2,1,1,981,kw_135,0,0.,0,0,0,"{Multivariate adaptive regression splines} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "moving_least_squares",8,2,1,1,991,kw_135,0,0.,0,0,0,"{Moving least squares} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "neural_network",8,3,1,1,997,kw_137,0,0.,0,0,0,"{Artificial neural network} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "polynomial",8,3,1,1,1015,kw_138,0,0.,0,0,0,"{Polynomial} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "radial_basis",8,4,1,1,1005,kw_139,
    "reuse_samples",8,3,3,0,1037,kw_140,0,0.,0,0,0,"{Sample reuse in global approximation} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "samples_file",11,0,4,0,1045,0,0.,0,0,0,"{File import of samples for global approximation builds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG",
    "use_gradients",8,0,5,0,1047,0,0.,0,0,0,"{Use of gradient data in global approximation builds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrG"}
## 7.1.3.144 static KeyWord kw_142  
[static]

Initial value:

```
{
  "additive",8,0,2,2,1089,0,0.,0.,0.,0,"[CHOOSE correction type]"),
  "combined",8,0,2,2,1093),
  "first_order",8,0,1,1,1085,0,0.,0.,0.,0,"[CHOOSE correction order]"),
  "multiplicative",8,0,2,2,1091),
  "second_order",8,0,1,1,1087),
  "zeroth_order",8,0,1,1,1083
}
```

## 7.1.3.145 static KeyWord kw_143  
[static]

Initial value:

```
{
  "correction",8,6,3,3,1081,kw_142,0.,0.,0.,0,0,"{Surrogate correction approach} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrH",
  "high_fidelity_model_pointer",11,0,2,2,1079,0,0.,0.,0.,0,"(Pointer to the high fidelity model specification) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrH",
  "low_fidelity_model_pointer",11,0,1,1,1077,0,0.,0.,0.,0,
}
```

## 7.1.3.146 GuiKeyWord kw_144[2]  
[static]

Initial value:

```
{
  "actual_model_pointer",11,0,2,2,1073,0,0.,0.,0.,0,"{Pointer to the truth model specification) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrMP",
  "taylor_series",8,0,1,1,1071,0,0.,0.,0.,0,"{Taylor series local approximation } http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrL"
}
```

## 7.1.3.147 GuiKeyWord kw_145[2]  
[static]

Initial value:

```
{
  "actual_model_pointer",11,0,2,2,1073,0,0.,0.,0.,0,"{Pointer to the truth model specification) http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrMP",
  "tana",8,0,1,1,1067,0,0.,0.,0.,0,"{Two-point adaptive nonlinear approximation } http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/ModelCommands.html#ModelSurrMP"
}
```

## 7.1.3.148 static KeyWord kw_146  
[static]

Initial value:
7.1 Dakota Namespace Reference

7.1.3.149 static KeyWord kw_147 [static]

Initial value:

```
{ "id_model", 11, 0, 1, 0, 953, 0, 0, 0, 0, 0, "Model set identifier" http://www.cs.sandia.gov/dakota/
 }{ "nested", 8, 2, 1, 1095, kw_128, 0, 0, 0, 0, 0, "Choose model type" },
{ "responses_pointer", 11, 0, 3, 0, 957, 0, 0, 0, 0, 0, "Responses set pointer" http://www.cs.sandia.gov/
 }{ "single", 8, 1, 4, 1, 959, kw_129, 0, 0, 0, 0, 0, "@" },
{ "surrogate", 8, 5, 4, 1, 963, kw_146 },
{ "variables_pointer", 11, 0, 2, 0, 955, 0, 0, 0, 0, 0, "Variables set pointer" http://www.cs.sandia.gov/ }
```

7.1.3.150 static KeyWord kw_148 [static]

Initial value:

```
{ "ignore_bounds", 8, 0, 1, 0, 1547, 0, 0, 0, 0, 0, "Ignore variable bounds" http://www.cs.sandia.gov/ }
```

7.1.3.151 static KeyWord kw_149 [static]

Initial value:

```
{ "central", 8, 0, 6, 0, 1555, 0, 0, 0, 0, 0, "Choose difference interval" },
{ "dakota", 8, 1, 4, 0, 1545, kw_148, 0, 0, 0, 0, 0, "@ Choose gradient source" },
{ "fd_gradient_step_size", 0x406, 0, 7, 0, 1556, 0, 0, 0, 0, 0.001 },
{ "fd_step_size", 0x40e, 0, 7, 0, 1557, 0, 0, 0, 0, 0, "Finite difference step size" http://www.cs.sandia.gov/ },
{ "forward", 8, 0, 6, 0, 1553, 0, 0, 0, 0, 0, "@" },
{ "id_analytic_gradients", 13, 0, 2, 2, 1539, 0, 0, 0, 0, 0, "Analytic derivatives function list" http://www.cs.sandia.gov/ },
{ "id_numerical_gradients", 13, 0, 1, 1, 1537, 0, 0, 0, 0, 0, "Numerical derivatives function list" http://www.cs.sandia.gov/ },
{ "interval_type", 8, 0, 5, 0, 1551, 0, 0, 0, 0, 0, "Interval type" http://www.cs.sandia.gov/dakota/ },
{ "method_source", 8, 0, 1, 0, 1543, 0, 0, 0, 0, 0, "Method source" http://www.cs.sandia.gov/dakota/ },
{ "vendor", 8, 0, 4, 0, 1549 }
```

7.1.3.152 static KeyWord kw_150 [static]

Initial value:
7.1.3.153  static KeyWord kw_151  

Initial value:

```
{  "damped",8,0,1,0,1593,0,0.,0.,0.,0,"{Numerical safeguarding of BFGS update} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/RespCommands.html#RespHessMixed"
}
```

7.1.3.154  static KeyWord kw_152  

Initial value:

```
{  "bfgs",8,1,1,1591,kw_151,0.,0.,0.,0,"[CHOOSE Hessian approx.]"},
  "sr1",8,0,1,1,1595
}
```

7.1.3.155  static KeyWord kw_153  

Initial value:

```
{  "central",8,0,2,0,1587,0,0.,0.,0.,0,"[CHOOSE difference interval]"},
  "forward",8,0,2,0,1585,0,0.,0.,0.,0,"[Forward difference]"},
  "id_analytic_hessians",13,0,4,0,1597,0,0.,0.,0.,0,"{Analytic Hessians function list} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/RespCommands.html#RespHessMixed"},
  "id_numerical_hessians",13,2,1,0,1581,kw_150,0.,0.,0.,0,"{Numerical Hessians function list} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/RespCommands.html#RespHessMixed"},
  "id_quasi_hessians",13,2,3,0,1589,kw_152,0.,0.,0.,0,"{Quasi Hessians function list} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/RespCommands.html#RespHessMixed"
}
```

7.1.3.156  static KeyWord kw_154  

Initial value:

```
{  "nonlinear_equality_scale_types",0x80f,0,2,0,1525,0,0.,0.,0.,0,"{Nonlinear equality scaling type} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/RespCommands.html#RespFnLS"},
  "nonlinear_equality_scales",0x80e,0,3,0,1527,0,0.,0.,0.,0,"{Nonlinear equality constraint scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/RespCommands.html#RespFnLS"},
  "nonlinear_equality_targets",14,0,1,0,1523,0,0.,0.,0.,0,"{Nonlinear equality constraint targets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/RespCommands.html#RespFnLS"
}
```
7.1.3.157  static KeyWord kw_155  [static]

Initial value:

```
{"nonlinear_inequality_lower_bounds",14,0,1,0,1513,0,0.,0.,0.,0.,"{Nonlinear inequality constraint lower bounds}"},
{"nonlinear_inequality_scale_types",0x80f,0,3,0,1517,0,0.,0.,0.,0.,"{Nonlinear inequality constraint scaling types}"},
{"nonlinear_inequality_scales",0x80e,0,4,0,1519,0,0.,0.,0.,0.,"{Nonlinear inequality constraint scales}"},
{"nonlinear_inequality_upper_bounds",14,0,2,0,1515,0,0.,0.,0.,0.,"{Nonlinear inequality constraint upper bounds}"}
```

7.1.3.158  static KeyWord kw_156  [static]

Initial value:

```
{"least_squares_data_file",11,0,1,0,1503,0,0.,0.,0.,0.,"{Least squares data source file}"},
{"least_squares_term_scale_types",0x80f,0,2,0,1505,0,0.,0.,0.,0.,"{Least squares term scaling types}"},
{"least_squares_term_scales",0x80e,0,3,0,1507,0,0.,0.,0.,0.,"{Least squares term scales}"},
{"least_squares_weights",14,0,4,0,1509,0,0.,0.,0.,0.,"{Least squares terms weightings}"},
{"num_nonlinear_equality_constraints",0x29,3,6,0,1521,kw_154,0.,0.,0.,0.,"{Number of nonlinear equality constraints}"},
{"num_nonlinear_inequality_constraints",0x29,4,5,0,1511,kw_155,0.,0.,0.,0.,"{Number of nonlinear inequality constraints}"}
```

7.1.3.159  static KeyWord kw_157  [static]

Initial value:

```
{"nonlinear_equality_scale_types",0x80f,0,2,0,1497,0,0.,0.,0.,0.,"{Nonlinear equality constraint scaling types}"},
{"nonlinear_equality_scales",0x80e,0,3,0,1499,0,0.,0.,0.,0.,"{Nonlinear equality constraint scales}"},
{"nonlinear_equality_targets",14,0,1,0,1495,0,0.,0.,0.,0.,"{Nonlinear equality constraint targets}"}
```

7.1.3.160  static KeyWord kw_158  [static]

Initial value:

```
{"nonlinear_inequality_lower_bounds",14,0,1,0,1485,0,0.,0.,0.,0.,"{Nonlinear inequality constraint lower bounds}"},
{"nonlinear_inequality_scale_types",0x80f,0,3,0,1489,0,0.,0.,0.,0.,"{Nonlinear inequality constraint scaling types}"},
{"nonlinear_inequality_scales",0x80e,0,4,0,1491,0,0.,0.,0.,0.,"{Nonlinear inequality constraint scales}"},
{"nonlinear_inequality_upper_bounds",14,0,2,0,1487,0,0.,0.,0.,0.,"{Nonlinear inequality constraint upper bounds}"}
```
7.1.3.161  static KeyWord kw_159 [static]

Initial value:

```json
{
    "multi_objective_weights", 14, 0, 3, 0, 1481, 0, 0., 0., 0., 0., 0, 
    "num_nonlinear_equality_constraints", 0x29, 3, 5, 0, 1493, kw_157, 0, 0., 0., 0., 0,
    "num_nonlinear_inequality_constraints", 0x29, 4, 4, 0, 1483, kw_158, 0, 0., 0., 0., 0,
    "objective_function_scale_types", 0x80f, 0, 1, 0, 1487, 0, 0., 0., 0., 0., 0,
    "objective_function_scales", 0x80e, 0, 2, 0, 1479, 0, 0., 0., 0., 0., 0,
}
```

7.1.3.162  GuiKeyWord kw_160[8] [static]

Initial value:

```json
{
    "central", 8, 0, 6, 0, 1555, 0, 0., 0., 0., 0., 0,
    "{CHOOSE difference interval}",
    "dakota", 8, 1, 4, 0, 1545, kw_148, 0, 0., 0., 0., 0,
    "{CHOOSE gradient source}",
    "fd_gradient_step_size", 0x406, 0, 7, 0, 1556, 0, 0., 0., 0., 0.001,
    "fd_step_size", 0x40e, 0, 7, 0, 1557, 0, 0., 0., 0., 0., 0.001,
    "forward", 8, 0, 6, 0, 1553, 0, 0., 0., 0., 0., 0,
    "interval_type", 8, 0, 5, 0, 1551, 0, 0., 0., 0., 0.
    "method_source", 8, 0, 3, 0, 1543, 0, 0., 0., 0., 0.
    "vendor", 8, 0, 4, 0, 1549
}
```

7.1.3.163  static KeyWord kw_161 [static]

Initial value:

```json
{
    "central", 8, 0, 2, 0, 1567, 0, 0., 0., 0., 0., 0,
    "{CHOOSE difference interval}",
    "fd_hessian_step_size", 6, 0, 1, 0, 1562,
    "fd_step_size", 14, 0, 1, 0, 1563, 0, 0., 0., 0., 0., 0,
    "forward", 8, 0, 2, 0, 1565, 0, 0., 0., 0., 0., 0,*
}
```

7.1.3.164  static KeyWord kw_162 [static]

Initial value:

```json
{
    "damped", 8, 0, 1, 0, 1573, 0, 0., 0., 0., 0., 0,
}
```
7.1.3.165 static KeyWord kw_163 [static]

Initial value:

```
{ "bfgs", 8, 1, 1, 1, 1571, kw_162, 0., 0., 0., 0., "[CHOOSE Hessian approx.]" },
{ "sr1", 8, 0, 1, 1, 1575 }
```

7.1.3.166 static KeyWord kw_164 [static]

Initial value:

```
{ "analytic_gradients", 8, 0, 4, 2, 1533, 0., 0., 0., 0., 0., 0., "[CHOOSE gradient type]"},
{ "analytic_hessians", 8, 0, 5, 3, 1577, 0., 0., 0., 0., 0., 0., "[CHOOSE Hessian type]"},
{ "descriptors", 15, 0, 1, 1573, 0., 0., 0., 0., 0., 0., "[Response labels] http://www.cs.sandia.gov/dakota/
  "mixed_gradients", 8, 10, 4, 2, 1535, kw_149, 0., 0., 0., 0., 0., 0., "[Mixed gradients] http://www.cs.sandia.gov/
  "no_gradients", 8, 0, 4, 2, 1531, 0., 0., 0., 0., 0., 0., "@"},
{ "no_hessians", 8, 0, 5, 3, 1559, 0., 0., 0., 0., 0., 0., "@"},
{ "num_least_squares_terms", 0x29, 6, 3, 1, 1501, kw_156, 0., 0., 0., 0., 0., 0., "[CHOOSE response type] (Least squares terms)
  "num_objective_functions", 0x29, 5, 3, 1, 1475, kw_159, 0., 0., 0., 0., 0., 0., "[Optimization] Number of object
  "num_response_functions", 0x29, 0, 3, 1, 1529, 0., 0., 0., 0., 0., 0., "[Generic responses] Number of response
  "numerical_gradients", 8, 8, 4, 2, 1541, kw_160, 0., 0., 0., 0., 0., 0., "[Numerical gradients] http://www.cs.sandia/
  "numerical_hessians", 8, 4, 5, 3, 1561, kw_161, 0., 0., 0., 0., 0., 0., "[Numerical Hessians] http://www.cs.sandia/
  "quasi_hessians", 8, 2, 5, 3, 1569, kw_163, 0., 0., 0., 0., 0., 0., "[Quasi Hessians] http://www.cs.sandia.gov/
  "response_descriptors", 7, 0, 2, 0, 1472 }
```

7.1.3.167 static KeyWord kw_165 [static]

Initial value:

```
{ "method_list", 15, 0, 1, 1, 33, 0., 0., 0., 0., 0., "[List of methods] http://www.cs.sandia.gov/dakota/
```

7.1.3.168 GuiKeyWord kw_166[3] [static]

Initial value:

```
{ "global_method_pointer", 11, 0, 1, 1, 25, 0., 0., 0., 0., 0., 0., "[Pointer to the global method specification]
  "local_method_pointer", 11, 0, 2, 2, 27, 0., 0., 0., 0., 0., 0., "[Pointer to the local method specification]
  "local_search_probability", 10, 0, 1, 0, 29, 0., 0., 0., 0., 0., 0., "[Probability of executing local searches] 
```
7.1.3.169 static KeyWord kw_167 [static]

Initial value:

```
{"method_list",15,0,2,1,21,0,0.,0.,0.,0,"List of methods http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratHybrid"},
{"num_solutions_transferred",9,0,1,0,19,0,0.,0.,0.,0,"Number of Solutions Transferred http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratHybrid"}
```

7.1.3.170 static KeyWord kw_168 [static]

Initial value:

```
{"collaborative",8,1,1,1,31,kw_165,0.,0.,0.,0,"Choose hybrid type Collaborative hybrid http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratHybrid"},
{"coupled",0,3,1,1,22,kw_166},
{"embedded",8,3,1,1,23,kw_166,0.,0.,0.,0,"Embedded hybrid http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratHybrid"},
{"sequential",8,2,1,1,17,kw_167,0.,0.,0.,0,"Sequential hybrid http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratHybrid"},
{"uncoupled",0,2,1,1,16,kw_167}
```

7.1.3.171 static KeyWord kw_169 [static]

Initial value:

```
{"seed",9,0,1,0,41,0,0.,0.,0.,0,"Seed for random starting points http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratMultiStart"}
```

7.1.3.172 static KeyWord kw_170 [static]

Initial value:

```
{"method_pointer",11,0,1,1,37,0,0.,0.,0.,0,"Method pointer http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratMultiStart"},
{"random_starts",9,1,2,0,39,kw_169,0.,0.,0.,0,"Number of random starting points http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratMultiStart"},
{"starting_points",14,0,3,0,43,0,0.,0.,0.,0,"List of user-specified starting points http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratMultiStart"}
```

7.1.3.173 static KeyWord kw_171 [static]

Initial value:

```
{"seed",9,0,1,0,51,0,0.,0.,0.,0,"Seed for random weighting sets http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratParetoSet"}
```
7.1 Dakota Namespace Reference

7.1.3.174 static KeyWord kw_172 [static]

Initial value:

```
{"method_pointer":11,0,1,1,47,0,0,0,0,0,"{Optimization method pointer} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratParetoSet"},
{"multi_objective_weight_sets":6,0,3,0,52},
{"opt_method_pointer":3,0,1,1,46},
{"random_weight_sets":9,1,2,0,49,kw_171,0,0,0,0,"{Number of random weighting sets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratParetoSet"},
{"weight_sets":14,0,3,0,53,0,0,0,0,0,"{List of user-specified weighting sets} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/StratCommands.html#StratParetoSet"}
```

7.1.3.175 GuiKeyWord kw_173[1] [static]

Initial value:

```
{"method_pointer":11,0,1,0,57,0,0,0,0,0,"{Method pointer} http://www.cs.sandia.gov/dakota/"}
```

7.1.3.176 static KeyWord kw_174 [static]

Initial value:

```
{"tabular_graphics_file":11,0,1,0,7,0,0,0,0,0,"{File name for tabular graphics data} http://www.cs.sandia.gov/dakota/"
```

7.1.3.177 static KeyWord kw_175 [static]

Initial value:

```
{"graphics":8,0,1,0,3,0,0,0,0,0,"{Graphics flag} http://www.cs.sandia.gov/dakota/licensing/"}
{"hybrid":8,5,6,1,15,kw_168,0,0,0,0,"{CHOOSE strategy type]Hybrid strategy} http://www.cs.sandia.gov"}
{"iterator_self_scheduling":8,0,4,0,11,0,0,0,0,0,"{Self-scheduling of iterator jobs] http://www.cs.sandia.gov"}
{"iterator_servers":9,0,3,0,9,0,0,0,0,0,"{Number of iterator servers} http://www.cs.sandia.gov"}
{"iterator_static_scheduling":8,0,5,0,13,0,0,0,0,0,"{Static scheduling of iterator jobs] http://www.cs.sandia.gov"}
{"multi_start":8,3,6,1,35,kw_170,0,0,0,0,"{Multi-start iteration strategy] http://www.cs.sandia.gov"}
{"pareto_set":8,5,6,1,45,kw_172,0,0,0,0,"{Pareto set optimization strategy] http://www.cs.sandia.gov"}
{"single_method":8,1,6,1,55,kw_173,0,0,0,0,"{Single method strategy] http://www.cs.sandia.gov"}
{"tabular_graphics_data":8,1,2,0,5,kw_174,0,0,0,0,"{Tabulation of graphics data] http://www.cs.sandia.gov"
```

7.1.3.178 static KeyWord kw_176 [static]

Initial value:

```
DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1.3.179 GuiKeyWord kw_177[3]  [static]

Initial value:

```
{"descriptors","15,0,3,0,1289,0,0,0,0,0","{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Binomial","binomial_uncertain"},
{"num_trials","13,0,2,2,1287,0,0,0,0,0","{binomial uncertain num_trials} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Binomial","binomial_uncertain"},
{"prob_per_trial","14,0,1,1,1285,0,0,0,0,0","{binomial uncertain prob_per_trial} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Binomial","binomial_uncertain"}
```

7.1.3.180 static KeyWord kw_178  [static]

Initial value:

```
{"cdv_descriptors","7,0,6,0,1126,0,0,0,0,0,0,0,0,0,0,"continuous_design"},
{"cdv_initial_point","6,0,1,0,1116,0,0,0,0,0,0,0,0,0,0,"continuous_design"},
{"cdv_lower_bounds","6,0,2,0,1118,0,0,0,0,0,0,0,0,0,0,"continuous_design"},
{"cdv_scales","0x807,0,4,0,1122,0,0,0,0,0,0,0,0,0,0,"continuous_design"},
{"cdv_upper_bounds","6,0,3,0,1120,0,0,0,0,0,0,0,0,0,0,"continuous_design"},
{"descriptors","15,0,6,0,1127,0,0,0,0,0,0,0,0,0,0,"descriptors" http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCDV"},
{"initial_point","14,0,1,0,1117,0,0,0,0,0,0,0,0,0,0,"{Initial point} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCDV"},
{"lower_bounds","14,0,2,0,1119,0,0,0,0,0,0,0,0,0,0,"{Lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCDV"},
{"scale_types","0x80f,0,4,0,1123,0,0,0,0,0,0,0,0,0,0,"{Scaling types} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCDV"},
{"scales","0x80e,0,5,0,1125,0,0,0,0,0,0,0,0,0,0,"{Scales} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCDV"},
{"upper_bounds","14,0,3,0,1121,0,0,0,0,0,0,0,0,0,0,"{Upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCDV"}
```

7.1.3.181 static KeyWord kw_179  [static]

Initial value:

```
{"csv_descriptors","7,0,4,0,1344,0,0,0,0,0,0,0,0,0,0,"continuous_state"},
{"csv_initial_state","6,0,1,0,1338,0,0,0,0,0,0,0,0,0,0,"continuous_state"},
{"csv_lower_bounds","6,0,2,0,1340,0,0,0,0,0,0,0,0,0,0,"continuous_state"},
{"csv_upper_bounds","6,0,3,0,1342,0,0,0,0,0,0,0,0,0,0,"continuous_state"},
{"descriptors","15,0,4,0,1345,0,0,0,0,0,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCSV"}
```
7.1 Dakota Namespace Reference

7.1.3.182 static KeyWord kw_180 [static]

Initial value:

```c
{"ddv_descriptors", 7, 0, 4, 0, 1136, 0, 0, 0, 0, 0, \"discrete_design_range\"},
{"ddv_initial_point", 5, 0, 1, 0, 1130, 0, 0, 0, 0, 0, \"discrete_design_range\"},
{"ddv_lower_bounds", 5, 0, 2, 0, 1132, 0, 0, 0, 0, 0, \"discrete_design_range\"},
{"ddv_upper_bounds", 5, 0, 3, 0, 1134, 0, 0, 0, 0, 0, \"discrete_design_range\"},
{"descriptors", 15, 0, 4, 0, 1137, 0, 0, 0, 0, 0, \"discrete_design_range\"},
{"initial_point", 13, 0, 1, 0, 1131, 0, 0, 0, 0, 0, \"discrete_design_range\"},
{"lower_bounds", 13, 0, 2, 0, 1133, 0, 0, 0, 0, 0, \"discrete_design_range\"},
{"upper_bounds", 13, 0, 3, 0, 1135, 0, 0, 0, 0, 0, \"discrete_design_range\"}
```

7.1.3.183 static KeyWord kw_181 [static]

Initial value:

```c
{"descriptors", 15, 0, 4, 0, 1147, 0, 0, 0, 0, 0, \"discrete_design_set_integer\"},
{"initial_point", 13, 0, 1, 0, 1141, 0, 0, 0, 0, 0, \"discrete_design_set_integer\"},
{"num_set_values", 13, 0, 2, 0, 1143, 0, 0, 0, 0, 0, \"discrete_design_set_integer\"},
{"set_values", 13, 0, 3, 1, 1145, 0, 0, 0, 0, 0, \"discrete_design_set_integer\"}
```

7.1.3.184 static KeyWord kw_182 [static]

Initial value:

```c
{"descriptors", 15, 0, 4, 0, 1157, 0, 0, 0, 0, 0, \"discrete_design_set_real\"},
{"initial_point", 14, 0, 1, 0, 1151, 0, 0, 0, 0, 0, \"discrete_design_set_real\"},
{"num_set_values", 13, 0, 2, 0, 1153, 0, 0, 0, 0, 0, \"discrete_design_set_real\"},
{"set_values", 14, 0, 3, 1, 1155, 0, 0, 0, 0, 0, \"discrete_design_set_real\"}
```

7.1.3.185 static KeyWord kw_183 [static]

Initial value:

```c
{"descriptors", 15, 0, 4, 0, 1355, 0, 0, 0, 0, 0, \"discrete_design_set_real\"}
```

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
7.1.3.186 static KeyWord kw_184

Initial value:

```
{"descriptors",15,0,4,0,1365,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSIV",0,"discrete_state_set_integer"},
{"initial_state",13,0,1,0,1359,0,0,0,0,0,"{Initial state} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSIV",0,"discrete_state_set_integer"},
{"num_set_values",13,0,2,0,1361,0,0,0,0,0,"{Number of values for each variable} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSIV",0,"discrete_state_set_integer"},
{"set_values",13,0,3,1,1363,0,0,0,0,0,"{Set values} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSIV"}
```

7.1.3.187 static KeyWord kw_185

Initial value:

```
{"descriptors",15,0,4,0,1375,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSRV",0,"discrete_state_set_real"},
{"initial_state",14,0,1,0,1369,0,0,0,0,0,"{Initial state} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSRV",0,"discrete_state_set_real"},
{"num_set_values",13,0,2,0,1371,0,0,0,0,0,"{Number of values for each variable} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSRV",0,"discrete_state_set_real"},
{"set_values",14,0,3,1,1373,0,0,0,0,0,"{Set values} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDSSRV"}
```

7.1.3.188 static KeyWord kw_186

Initial value:

```
{"betas",14,0,1,1,1217,0,0,0,0,0,"{exponential uncertain betas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Exponential",0,"exponential_uncertain"},
{"descriptors",15,0,2,0,1219,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Exponential",0,"exponential_uncertain"},
{"euv_betas",6,0,1,1,1216,0,0,0,0,0,"exponential_uncertain"},
{"euv_descriptors",7,0,2,0,1218,0,0,0,0,0,"exponential_uncertain"}
```

7.1.3.189 static KeyWord kw_187

Initial value:

```
{"alphas",14,0,1,1,1251,0,0,0,0,0,"{frechet uncertain alphas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Frechet",0,"frechet_uncertain"},
{"euv_alphas",6,0,1,1,1248,0,0,0,0,0,"frechet_uncertain"},
{"euv_descriptors",7,0,2,0,1250,0,0,0,0,0,"frechet_uncertain"}
```
7.1 Dakota Namespace Reference

7.1.3.190 static KeyWord kw_188 [static]

Initial value:

```
{"alphas",14,0,1,1,1235,0,0.,0.,0.,0,"{gamma uncertain alphas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Gamma",0,"gamma_uncertain"},
{"betas",14,0,2,2,1237,0,0.,0.,0.,0,"{gamma uncertain betas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Gamma",0,"gamma_uncertain"},
{"descriptors",15,0,3,0,1239,0,0.,0.,0.,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Gamma",0,"gamma_uncertain"},
{"gauv_alphas",6,0,1,1,1234,0,0.,0.,0.,0,"gamma_uncertain"},
{"gauv_betas",6,0,2,2,1236,0,0.,0.,0.,0,"gamma_uncertain"},
{"gauv_descriptors",7,0,3,0,1238,0,0.,0.,0.,0,"gamma_uncertain"}
```

7.1.3.191 static KeyWord kw_189 [static]

Initial value:

```
{"descriptors",15,0,2,0,1303,0,0.,0.,0.,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Geometric",0,"geometric_uncertain"},
{"prob_per_trial",14,0,1,1,1301,0,0.,0.,0.,0,"{geometric uncertain prob_per_trial} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Geometric",0,"geometric_uncertain"}
```

7.1.3.192 static KeyWord kw_190 [static]

Initial value:

```
{"alphas",14,0,1,1,1243,0,0.,0.,0.,0,"{gumbel uncertain alphas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Gumbel",0,"gumbel_uncertain"},
{"betas",14,0,2,2,1245,0,0.,0.,0.,0,"{gumbel uncertain betas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Gumbel",0,"gumbel_uncertain"},
{"descriptors",15,0,3,0,1247,0,0.,0.,0.,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Gumbel",0,"gumbel_uncertain"},
{"guuv_alphas",6,0,1,1,1242,0,0.,0.,0.,0,"gumbel_uncertain"},
{"guuv_betas",6,0,2,2,1244,0,0.,0.,0.,0,"gumbel_uncertain"},
{"guuv_descriptors",7,0,3,0,1246,0,0.,0.,0.,0,"gumbel_uncertain"}
```

7.1.3.193 static KeyWord kw_191 [static]

Initial value:

```
{"abscissas",14,0,2,1,1269,0,0.,0.,0.,0,"{sets of abscissas for bin-based histogram variables} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Bin_Histogram",0,"bin_based_histogram_uncertain"}
```
7.1.3.194 static KeyWord kw_192 [static]
Initial value:

{ "abscissas",14,0,2,1,1319,0,0,0,0,0,"{sets of abscissas for point-based histogram variables} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Point_Histogram" },
{ "counts",14,0,3,2,1321,0,0,0,0,0,"{sets of counts for point-based histogram variables} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Point_Histogram" },
{ "descriptors",15,0,4,0,1317,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Point_Histogram" },
{ "huv_point_abscissas",6,0,3,2,1322 },
{ "huv_point_counts",6,0,3,2,1323 },
{ "huv_point_descriptors",7,0,4,0,1324,0,0,0,0,0,"histogram point uncertain" },
{ "huv_point_ordinates",6,0,3,2,1325 },
{ "huv_num_point_pairs",5,0,1,0,1326,0,0,0,0,0,"histogram point uncertain" },
{ "num_pairs",13,0,1,0,1327,0,0,0,0,0,"{key to apportionment among point-based histogram variables} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Point_Histogram" }

7.1.3.195 GuiKeyWord kw_193[4] [static]
Initial value:

{ "descriptors",15,0,4,0,1313,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Hypergeometric" },
{ "num_drawn",13,0,3,3,1311,0,0,0,0,0,"{hypergeometric uncertain num drawn} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Hypergeometric" },
{ "selected_population",13,0,2,2,1309,0,0,0,0,0,"{hypergeometric uncertain selected population} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Hypergeometric" },
{ "total_population",13,0,1,1,1307,0,0,0,0,0,"{hypergeometric uncertain total population} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Hypergeometric" }

7.1.3.196 GuiKeyWord kw_194[8] [static]
Initial value:

{ "descriptors",15,0,4,0,1335,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCEUV_Interval" },
{ "interval_bounds",14,0,3,2,1333,0,0,0,0,0,"{bounds per interval} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCEUV_Interval" },
{ "interval_probs",14,0,2,1,1331,0,0,0,0,0,"{basic probability assignments per interval} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCEUV_Interval" },
{ "iuv_descriptors",7,0,4,0,1334,0,0,0,0,0,"{interval uncertain} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCEUV_Interval" },
{ "iuv_interval_bounds",6,0,3,2,1332 },
{ "iuv_interval_probs",6,0,2,1,1330 },
{ "iuv_num_intervals",5,0,1,0,1328,0,0,0,0,0,"{number of intervals defined for each interval variable} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCEUV_Interval" }
7.1 Dakota Namespace Reference

7.1.3.197 static KeyWord kw_195 [static]
Initial value:

```
{"lnuv_zetas",6,0,1,1,1174,0,0,0,0,0,0,0,0,"lognormal_uncertain"},
{"zetas",14,0,1,1,1175,0,0,0,0,0,0,"{lognormal uncertain zetas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"}
```

7.1.3.198 GuiKeyWord kw_196[4] [static]
Initial value:

```
{"error_factors",14,0,1,1,1181,0,0,0,0,0,0,"[CHOOSE variance spec.]{lognormal uncertain error factors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"},
{"lnuv_error_factors",6,0,1,1,1180,0,0,0,0,0,0,"lognormal_uncertain"},
{"lnuv_std_deviations",6,0,1,1,1178,0,0,0,0,0,0,"lognormal_uncertain"},
{"std_deviations",14,0,1,1,1179,0,0,0,0,0,0,"{lognormal uncertain standard deviations} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"}
```

7.1.3.199 static KeyWord kw_197 [static]
Initial value:

```
{"descriptors",15,0,4,0,1187,0,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"},
{"lambdas",14,2,1,1,1173,kw_195,0,0,0,0,0,"[CHOOSE characterization]{lognormal uncertain lambda lambdas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"},
{"lnuv_descriptors",7,0,4,0,1186,0,0,0,0,0,0,"lognormal_uncertain"},
{"lnuv_lambdas",6,2,1,1,1172,kw_195,0,0,0,0,0,0,"lognormal_uncertain"},
{"lnuv_lower_bounds",6,0,2,0,1182,0,0,0,0,0,0,"lognormal_uncertain"},
{"lnuv_means",6,4,1,1,1176,kw_196,0,0,0,0,0,0,"lognormal_uncertain"},
{"lnuv_upper_bounds",6,0,3,0,1184,0,0,0,0,0,0,"lognormal_uncertain"},
{"lower_bounds",14,0,2,0,1183,0,0,0,0,0,0,"{Distribution lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"},
{"means",14,4,1,1,1177,kw_196,0,0,0,0,0,0,"{lognormal uncertain means} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"},
{"upper_bounds",14,0,3,0,1185,0,0,0,0,0,0,"{Distribution upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Lognormal"}
```

7.1.3.200 GuiKeyWord kw_198[6] [static]
Initial value:

```
{"descriptors",15,0,3,0,1203,0,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Loguniform"},
{"lower_bounds",14,0,1,1,1199,0,0,0,0,0,0,"{Distribution lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Loguniform"},
{"luuv_descriptors",7,0,3,0,1202,0,0,0,0,0,0,"{loguniform uncertain descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Loguniform"},
{"luuv_lambdas",6,0,1,1,1198,0,0,0,0,0,0,"loguniform_uncertain"},
{"luuv_lower_bounds",6,0,2,2,1200,0,0,0,0,0,0,"loguniform_uncertain"},
{"upper_bounds",14,0,2,2,1201,0,0,0,0,0,0,"{Distribution upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Loguniform"}
```
7.1.3.201 static KeyWord kw_199 [static]

Initial value:

```
{ "descriptors", 15, 0, 3, 0, 1297, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Negative_Binomial", 0, "negative_binomial_uncertain" },
{ "num_trials", 13, 0, 2, 2, 1295, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{negative binomial uncertain success num_trials} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Negative_Binomial", 0, "negative_binomial_uncertain" },
{ "prob_per_trial", 14, 0, 1, 1, 1293, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{negative binomial uncertain success prob_per_trial} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Negative_Binomial", 0, "negative_binomial_uncertain" }
```

7.1.3.202 static KeyWord kw_200 [static]

Initial value:

```
{ "descriptors", 15, 0, 5, 0, 1169, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Normal", 0, "normal_uncertain" },
{ "lower_bounds", 14, 0, 3, 0, 1165, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Distribution lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Normal", 0, "normal_uncertain" },
{ "means", 14, 0, 1, 1, 1161, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{normal uncertain means} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Normal", 0, "normal_uncertain" },
{ "nuv_descriptors", 7, 0, 5, 0, 1168, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "normal_uncertain" },
{ "nuv_lower_bounds", 6, 0, 3, 0, 1164, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "normal_uncertain" },
{ "nuv_means", 6, 0, 1, 1, 1160, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "normal_uncertain" },
{ "nuv_std_deviations", 6, 0, 2, 2, 1162, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "normal_uncertain" },
{ "nuv_upper_bounds", 6, 0, 4, 0, 1166, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "normal_uncertain" },
{ "std_deviations", 14, 0, 2, 2, 1163, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "normal uncertain standard deviations} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Normal", 0, "normal_uncertain" },
{ "upper_bounds", 14, 0, 4, 0, 1167, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Distribution upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Normal", 0, "normal_uncertain" }
```

7.1.3.203 static KeyWord kw_201 [static]

Initial value:

```
{ "descriptors", 15, 0, 2, 0, 1281, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Poisson", 0, "poisson_uncertain" },
{ "lambdas", 14, 0, 1, 1, 1279, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{poisson uncertain lambdas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarDAUV_Poisson", 0, "poisson_uncertain" }
```

7.1.3.204 static KeyWord kw_202 [static]

Initial value:

```
{ "descriptors", 15, 0, 4, 0, 1213, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Triangular", 0, "triangular_uncertain" },
{ "lower_bounds", 14, 0, 2, 2, 1209, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Distribution lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Triangular", 0, "triangular_uncertain" },
{ "modes", 14, 0, 1, 1, 1207, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{triangular uncertain modes} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Triangular", 0, "triangular_uncertain" },
{ "nuv_descriptors", 7, 0, 4, 0, 1212, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "triangular_uncertain" },
{ "nuv_lower_bounds", 6, 0, 2, 2, 1208, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "triangular_uncertain" },
{ "nuv_modes", 6, 0, 1, 1, 1206, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "triangular_uncertain" },
{ "nuv_upper_bounds", 6, 0, 3, 3, 1210, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "triangular_uncertain" },
{ "upper_bounds", 14, 0, 5, 3, 1211, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "{Distribution upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Triangular", 0, "triangular_uncertain" }
```

7.1.3.205  static KeyWord kw_203  [static]

Initial value:

```
{
  "descriptors",15,0,3,0,195,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Uniform",
  "lower_bounds",14,0,1,1,191,0,0,0,0,0,"{Distribution lower bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Uniform",
  "upper_bounds",14,0,2,2,193,0,0,0,0,0,"{Distribution upper bounds} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Uniform",
  "uuv_descriptors",7,0,3,0,1194,0,0,0,0,0,"uniform_uncertain",
  "uuv_lower_bounds",6,0,1,1,1190,0,0,0,0,0,"uniform_uncertain",
  "uuv_upper_bounds",6,0,2,2,1192,0,0,0,0,0,"uniform_uncertain"
}
```

7.1.3.206  static KeyWord kw_204  [static]

Initial value:

```
{
  "alphas",14,0,1,1,1259,0,0,0,0,0,"{weibull uncertain alphas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Weibull",
  "betas",14,0,2,2,1261,0,0,0,0,0,"{weibull uncertain betas} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Weibull",
  "descriptors",15,0,3,0,1263,0,0,0,0,0,"{Descriptors} http://www.cs.sandia.gov/dakota/licensing/votd/html-ref/VarCommands.html#VarCAUV_Weibull",
  "wuv_alphas",6,0,1,1,1258,0,0,0,0,0,"weibull_uncertain",
  "wuv_betas",6,0,2,2,1260,0,0,0,0,0,"weibull_uncertain",
  "wuv_descriptors",7,0,3,0,1262,0,0,0,0,0,"weibull_uncertain"
}
```

7.1.3.207  static KeyWord kw_206  [static]

Initial value:

```
{
  "interface",0x308,10,5,5,1377,kw_9,0,0,0,0,0,"{Interface} An interface specifies how function evaluations will be performed, possibly through the use of an interface file. http://www.cs.sandia.gov/DAKOTA/licensing/votd/html-ref/InterfCommands.html",
  "method",0x308,60,2,2,59,kw_125,0,0,0,0,0,"{Method} A method specifies the name and controls of an iterative procedure, e.g., quantification, optimization method. http://www.cs.sandia.gov/DAKOTA/licensing/votd/html-ref/MethodCommands.html",
  "model",8,6,3,3,951,kw_147,0,0,0,0,0,"{Model} A model consists of a model type and maps specified variables through an interface to generate responses. http://www.cs.sandia.gov/DAKOTA/licensing/votd/html-ref/ModelCommands.html",
  "responses",0x308,15,6,6,1469,kw_164,0,0,0,0,0,"{Responses} A responses object specifies the data that can be returned from a function evaluation. http://www.cs.sandia.gov/DAKOTA/licensing/votd/html-ref/RespCommands.html",
  "strategy",0x108,9,1,1,kw_175,0,0,0,0,0,"{Strategy} The strategy specifies the top level technique which will govern the iterated solution of the problem of interest. http://www.cs.sandia.gov/DAKOTA/licensing/votd/html-ref/StratCommands.html",
  "variables",0x308,29,4,4,1111,kw_205,0,0,0,0,0,"{Variables} A variables object specifies the parameter set to be iterated by a particular method. http://www.cs.sandia.gov/DAKOTA/licensing/votd/html-ref/VarCommands.html"
}
```

7.1.3.208  KeyWord kw_1[3]  [static]

Initial value:

```
{
  "active_set_vector",8,0,1,0,0,0,0,0,0,N_ifm(false,activeSetVectorFlag)
}
```

799 distinct keywords (plus 89 aliases)
7.1.3.209  KeyWord kw_2[1]  [static]
Initial value:

{
    "processors_per_analysis", 9, 0, 1, 0, 0, 0., 0, 0, N_ifm(int, procsPerAnalysis)
}

7.1.3.210  KeyWord kw_3[4]  [static]
Initial value:

{
    "abort", 8, 0, 1, 1, 0, 0., 0, 0, N_ifm(lit, failAction_abort),
    "continuation", 8, 0, 1, 1, 0, 0., 0, 0, N_ifm(lit, failAction_continuation),
    "recover", 14, 0, 1, 1, 0, 0., 0, 0, N_ifm(Rlit, 3failAction_recover),
    "retry", 9, 0, 1, 1, 0, 0., 0, 0, N_ifm(lit, 3failAction_retry)
}

7.1.3.211  KeyWord kw_4[2]  [static]
Initial value:

{
    "copy", 8, 0, 1, 0, 0., 0, 0, 0, N_ifm(true, templateCopy),
    "replace", 8, 0, 2, 0, 0., 0, 0, 0, N_ifm(true, templateReplace)
}

7.1.3.212  KeyWord kw_5[7]  [static]
Initial value:

{
    "dir_save", 0, 0, 3, 0, 0, 0., 0, 2, N_ifm(true, dirSave),
    "dir_tag", 0, 0, 2, 0, 0., 0, 0, 2, N_ifm(true, dirTag),
    "directory_save", 8, 0, 3, 0, 0, 0., 0, 0, N_ifm(true, dirSave),
    "directory_tag", 8, 0, 2, 0, 0., 0, 0, 0, N_ifm(true, dirTag),
    "named", 11, 0, 1, 0, 0., 0, 0, 0, N_ifm(str, workDir),
    "template_directory", 11, 2, 4, 0, kw_4, 0., 0, 0, N_ifm(str, templateDir),
    "template_files", 15, 2, 4, 0, kw_4, 0., 0, 0, N_ifm(strL, templateFiles)
}

7.1.3.213  KeyWord kw_6[7]  [static]
Initial value:


7.1 Dakota Namespace Reference

```json
{
  "aprepro", 8, 0, 4, 0, 0, 0, 0, 0, N_ifm(true, apreproFlag),
  "file_save", 8, 0, 6, 0, 0, 0, 0, 0, N_ifm(true, fileSaveFlag),
  "file_tag", 8, 0, 5, 0, 0, 0, 0, 0, N_ifm(true, fileTagFlag),
  "parameters_file", 11, 0, 1, 0, 0, 0, 0, 0, N_ifm(str, parametersFile),
  "results_file", 11, 0, 2, 0, 0, 0, 0, 0, N_ifm(str, resultsFile),
  "verbatim", 8, 0, 3, 0, 0, 0, 0, 0, N_ifm(true, verbatimFlag),
  "work_directory", 8, 7, 7, 0, kw_5, 0, 0, 0, 0, N_ifm(true, useWorkdir)
}
```

7.1.3.214 KeyWord kw_7[9] [static]

Initial value:

```json
{
  "analysis_components", 15, 0, 1, 0, 0, 0, 0, 0, N_ifm(str2D, analysisComponents),
  "deactivate", 8, 3, 6, 0, kw_1, 0, 0, 0, 0, 0, 0, 0, 0, N_ifm(bool, deactivateFlag),
  "direct", 8, 1, 4, 1, kw_2, 0, 0, 0, 0, N_ifm(lit, interfaceType_direct),
  "failure_capture", 8, 4, 5, 0, kw_3, 0, 0, 0, 0, 0, 0, 0, 0, 0, N_ifm(lit, interfaceType_failure_capture),
  "fork", 8, 7, 4, 1, kw_6, 0, 0, 0, 0, N_ifm(lit, interfaceType_fork),
  "grid", 8, 0, 4, 1, 0, 0, 0, 0, 0, N_ifm(lit, interfaceType_grid),
  "input_filter", 11, 0, 2, 0, 0, 0, 0, 0, 0, N_ifm(str, inputFilter),
  "output_filter", 11, 0, 3, 0, 0, 0, 0, 0, 0, N_ifm(str, outputFilter),
  "system", 8, 7, 4, 1, kw_6, 0, 0, 0, 0, N_ifm(lit, interfaceType_system)
}
```

7.1.3.215 KeyWord kw_8[4] [static]

Initial value:

```json
{
  "analysis_concurrency", 9, 0, 3, 0, 0, 0, 0, 0, N_ifm(int, asynchLocalAnalysisConcurrency),
  "evaluation_concurrency", 9, 0, 1, 0, 0, 0, 0, 0, N_ifm(int, asynchLocalEvalConcurrency),
  "local_evaluation_self_scheduling", 8, 0, 2, 0, 0, 0, 0, 0, N_ifm(lit, asynchLocalEvalScheduling_self),
  "local_evaluation_static_scheduling", 8, 0, 2, 0, 0, 0, 0, 0, 0, N_ifm(lit, asynchLocalEvalScheduling_static)
}
```

7.1.3.216 KeyWord kw_9[10] [static]

Initial value:

```json
{
  "algebraic_mappings", 11, 0, 2, 0, 0, 0, 0, 0, 0, N_ifm(str, algebraicMappings),
  "analysis_drivers", 15, 9, 3, 0, kw_7, 0, 0, 0, 0, N_ifm(strL, analysisDrivers),
  "analysis_self_scheduling", 8, 0, 8, 0, 0, 0, 0, 0, 0, N_ifm(lit, analysisScheduling_self),
  "analysis_servers", 9, 0, 7, 0, 0, 0, 0, 0, 0, N_ifm(int, analysisServers),
  "analysis_static_scheduling", 8, 0, 8, 0, 0, 0, 0, 0, 0, N_ifm(lit, analysisScheduling_static),
  "asynchronous", 8, 4, 4, 0, kw_8, 0, 0, 0, 0, N_ifm(lit, interfaceSynchronization_asynchronous),
  "evaluation_self_scheduling", 8, 0, 6, 0, 0, 0, 0, 0, 0, N_ifm(lit, evalScheduling_self),
  "evaluation_servers", 9, 0, 5, 0, 0, 0, 0, 0, 0, N_ifm(int, evalServers),
  "evaluation_static_scheduling", 8, 0, 6, 0, 0, 0, 0, 0, 0, N_ifm(lit, evalScheduling_static),
  "id_interface", 11, 0, 1, 0, 0, 0, 0, 0, 0, N_ifm(str, idInterface)
}
```
7.1.3.217  KeyWord kw_18[2]  [static]

Initial value:

```c
{
   {"all_dimensions",8,0,1,1,0,0.,0.,0,N_mdm(lit,boxDivision_all_dimensions)},
   {"major_dimension",8,0,1,1,0,0.,0.,0,N_mdm(lit,boxDivision_major_dimension)}
}
```

7.1.3.218  KeyWord kw_33[1]  [static]

Initial value:

```c
{
   {"seed",9,0,1,0,0.,0.,0.,0,N_mdm(pint,randomSeed)}
}
```

7.1.3.219  KeyWord kw_41[1]  [static]

Initial value:

```c
{
   {"list_of_points",14,0,1,1,0,0.,0.,0,N_mdm(RealDL,listOfPoints)}
}
```

7.1.3.220  KeyWord kw_58[2]  [static]

Initial value:

```c
{
   {"mt19937",8,0,1,1,0,0.,0.,0,N_mdm(lit,rngName_mt19937)},
   {"rnum2",8,0,1,1,0,0.,0.,0,N_mdm(lit,rngName_rnum2)}
}
```

7.1.3.221  KeyWord kw_60[2]  [static]

Initial value:

```c
{
   {"complementary",8,0,1,1,0,0.,0.,0,N_mdm(lit,distributionType_complementary)},
   {"cumulative",8,0,1,1,0,0.,0.,0,N_mdm(lit,distributionType_cumulative)}
}
```
7.1 Dakota Namespace Reference

7.1.3.222  KeyWord kw_68[2]  [static]

Initial value:

```
{ "mt19937", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, rngName_mt19937) },
{ "rnum2", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, rngName_rnum2) }
```

7.1.3.223  KeyWord kw_74[2]  [static]

Initial value:

```
{ "gen_reliabilities", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, responseLevelMappingType_gen_reliabilities) },
{ "probabilities", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, responseLevelMappingType_probabilities) }
```

7.1.3.224  KeyWord kw_78[2]  [static]

Initial value:

```
{ "gen_reliabilities", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, responseLevelMappingType_gen_reliabilities) },
{ "probabilities", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, responseLevelMappingType_probabilities) }
```

7.1.3.225  KeyWord kw_89[3]  [static]

Initial value:

```
{ "first_order", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, reliabilityIntegration_first_order) },
{ "refinement", 8, 5, 2, 0, kw_88, 0, 0, 0 },
{ "second_order", 8, 0, 1, 1, 0, 0, 0, 0, N_mdm(lit, reliabilityIntegration_second_order) }
```

7.1.3.226  KeyWord kw_95[1]  [static]

Initial value:

```
{ "num_reliability_levels", 13, 0, 1, 0, 0, 0, 0, 0, N_mdm(num_resplevs, reliabilityLevels) }
```
7.1.3.227  **KeyWord kw_100[2]**  [static]

Initial value:

```c
{
    "expansion_order", 13, 0, 1, 1, 0, 0., 0., 0., N_mdm(ushintL, expansionOrder),
    "expansion_terms", 9, 0, 1, 1, 0, 0., 0., 0., N_mdm(int, expansionTerms)
}
```

7.1.3.228  **KeyWord kw_107[4]**  [static]

Initial value:

```c
{
    "incremental_lhs", 8, 1, 1, 1, kw_106, 0., 0., 0., N_mdm(lit, sampleType_incremental_lhs),
    "incremental_random", 8, 1, 1, kw_106, 0., 0., 0., N_mdm(lit, sampleType_incremental_random),
    "lhs", 8, 0, 1, 1, 0, 0., 0., 0., N_mdm(lit, sampleType_lhs),
    "random", 8, 0, 1, 1, 0, 0., 0., 0., N_mdm(lit, sampleType_random)
}
```

7.1.3.229  **KeyWord kw_114[2]**  [static]

Initial value:

```c
{
    "gradient_tolerance", 10, 0, 2, 0, 0., 0., 0., 0., N_mdm(Real, gradientTolerance),
    "max_step", 10, 0, 1, 0, 0., 0., 0., 0., N_mdm(Real, maxStep)
}
```

7.1.3.230  **KeyWord kw_128[2]**  [static]

Initial value:

```c
{
    "filter", 8, 0, 1, 1, 0, 0., 0., 0., N_mdm(slit, surrBasedLocalAcceptLogic_FILTER),
    "tr_ratio", 8, 0, 1, 1, 0, 0., 0., 0., N_mdm(slit, surrBasedLocalAcceptLogic_TR_RATIO)
}
```

7.1.3.231  **KeyWord kw_144[2]**  [static]

Initial value:

```c
{
    "cubic", 8, 0, 1, 1, 0, 0., 0., 0., N_mom(lit, marsInterpolation_cubic),
    "linear", 8, 0, 1, 1, 0, 0., 0., 0., N_mom(lit, marsInterpolation_linear)
}
```
### 7.1.3.232 KeyWord kw_145[2] [static]

**Initial value:**

```c
{
    "interpolation", 8, 2, 2, kw_144, 0, 0, 0, 0,
    "max_bases", 9, 0, 1, 0, 0, 0, 0, N_mom(shint, marsMaxBases)
}
```

### 7.1.3.233 KeyWord kw_160[8] [static]

**Initial value:**

```c
{
    "central", 8, 0, 0, 0, 0, 0, 0, 0,
    "dakota", 8, 1, 0, kw_159, 0, 0, 0, 0,
    "fd_gradient_step_size", 0x406, 0, 0, 0, 0, 0, 0, 1,
    "fd_step_size", 0x40e, 0, 0, 0, 0, 0, 0, 0,
    "forward", 8, 0, 0, 0, 0, 0, 0, 0,
    "interval_type", 8, 0, 3, 0, 0, 0, 0, 0,
    "method_source", 8, 0, 1, 0, 0, 0, 0, 0,
    "vendor", 8, 0, 2, 0, 0, 0, 0, 0,
}
```

### 7.1.3.234 KeyWord kw_166[3] [static]

**Initial value:**

```c
{
    "damped", 8, 0, 1, 0, 0, 0, 0, 0,
}
```

### 7.1.3.235 KeyWord kw_173[1] [static]

**Initial value:**

```c
{
    "nonlinear_equality_scale_types", 0x80f, 0, 2, 0, 0, 0, 0, 0,
    "nonlinear_equality_scales", 0x80e, 0, 3, 0, 0, 0, 0, 0,
    "nonlinear_equality_targets", 14, 0, 1, 0, 0, 0, 0, 0,
}
```

### 7.1.3.236 KeyWord kw_177[3] [static]

**Initial value:**

```c
{
    "global_method_pointer", 11, 0, 1, 1, 0, 0, 0, 0,
    "local_method_pointer", 11, 0, 2, 2, 0, 0, 0, 0,
    "local_search_probability", 10, 0, 3, 0, 0, 0, 0, 0,
}
```
7.1.3.237 KeyWord kw_193[4] [static]

Initial value:

```
{
    "descriptors",15,0,4,0,0,0,0,0,0,N_vam(strL,discreteDesignSetRealLabels)),
    "initial_point",14,0,1,0,0,0,0,0,0,N_vam(RealLd,discreteDesignSetRealVars)),
    "num_set_values",13,0,2,0,0,0,0,0,0,N_vam(vil,Var_Info_ndsvr)),
    "set_values",14,0,3,1,0,0,0,0,0,N_vam(vrl,Var_Info_dsvr))
}
```

7.1.3.238 KeyWord kw_194[8] [static]

Initial value:

```
{
    "descriptors",15,0,4,0,0,0,0,0,0,N_vam(strL,discreteStateRangeLabels)),
    "dsv_descriptors",7,0,4,0,0,0,0,0,3,N_vam(strL,discreteStateRangeLabels)),
    "dsv_initial_state",5,0,2,0,0,0,0,0,3,N_vam(intDL,discreteStateRangeVars)),
    "dsv_lower_bounds",5,0,3,0,0,0,0,0,3,N_vam(intDL,discreteStateRangeLowerBnds)),
    "dsv_upper_bounds",5,0,3,0,0,0,0,0,3,N_vam(intDL,discreteStateRangeUpperBnds)),
    "initial_state",13,0,1,0,0,0,0,0,0,N_vam(intDL,discreteStateRangeVars)),
    "lower_bounds",13,0,2,0,0,0,0,0,0,N_vam(intDL,discreteStateRangeLowerBnds)),
    "upperBounds",13,0,3,0,0,0,0,0,0,N_vam(intDL,discreteStateRangeUpperBnds))
}
```

7.1.3.239 KeyWord kw_196[4] [static]

Initial value:

```
{
    "descriptors",15,0,4,0,0,0,0,0,0,N_vam(strL,discreteStateSetRealLabels)),
    "initial_state",14,0,1,0,0,0,0,0,0,N_vam(RealLd,discreteStateSetRealVars)),
    "num_set_values",13,0,2,0,0,0,0,0,0,N_vam(vil,Var_Info_nssvr)),
    "set_values",14,0,3,1,0,0,0,0,0,N_vam(vrl,Var_Info_ssvr))
}
```

7.1.3.240 KeyWord kw_198[6] [static]

Initial value:

```
{
    "alphas",14,0,1,1,0,0,0,0,0,N_vam(Reallb,frechetUncAlphas)),
    "betas",14,0,2,2,0,0,0,0,0,N_vam(RealLd,frechetUncBetas)),
    "descriptors",15,0,3,0,0,0,0,0,0,N_vae(caulbl,CAUVar_frechet)),
    "fuv_alphas",6,0,1,1,0,0,0,0,0,0,N_vam(Reallb,frechetUncAlphas)),
    "fuv_betas",6,0,2,2,0,0,0,0,0,0,N_vam(RealLd,frechetUncBetas)),
    "fuv_descriptors",7,0,3,0,0,0,0,0,0,0,N_vae(caulbl,CAUVar_frechet))
}
```
7.1.3.241  KeyWord kw_207[4]  [static]

Initial value:

```
{
    "error_factors",14,0,1,1,0,0.,0.,0.,0,N_vam(RealLb,lognormalUncErrFacts)),
    "lnuv_error_factors",6,0,1,1,0,0.,0.,-1,N_vam(RealLb,lognormalUncErrFacts)),
    "lnuv_std_deviations",6,0,1,1,0,0.,0.,1,N_vam(RealLb,lognormalUncStdDevs)),
    "std_deviations",14,0,1,1,0,0.,0.,0,N_vam(RealLb,lognormalUncStdDevs)
}
```

7.1.3.242  KeyWord kw_208[10]  [static]

Initial value:

```
{
    "descriptors",15,0,4,0,0,0.,0.,0.,0,N_vae(caulbl,CAUVar_lognormal)),
    "lambdas",14,2,1,1,kw_206,0,0,0,0,N_vam(RealLd,lognormalUncLambdas)),
    "lnuv_descriptors",6,4,1,1,kw_207,0,0,0,0,N_vae(caulbl,CAUVar_lognormal)),
    "lnuv_lambdas",6,2,1,1,kw_206,0,0,0,0,N_vam(RealLd,lognormalUncLambdas)),
    "lnuv_lower_bounds",6,0,2,0,0,0,0,0,3,N_vam(RealLb,lognormalUncLowerBnds)),
    "lnuv_means",6,4,1,1,kw_207,0,0,0,0,3,N_vam(RealLb,lognormalUncMeans)),
    "lnuv_upper_bounds",6,0,3,0,0,0,0,3,N_vam(RealUb,lognormalUncUpperBnds)),
    "lower_bounds",14,0,2,0,0,0,0,0,0,N_vam(RealLb,lognormalUncLowerBnds)),
    "means",14,4,1,1,kw_207,0,0,0,0,N_vam(RealLb,lognormalUncMeans)),
    "upper_bounds",14,0,3,0,0,0,0,0,0,N_vam(RealUb,lognormalUncUpperBnds)
}
```

7.1.3.243  KeyWord kw_209[6]  [static]

Initial value:

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

7.1.3.244  KeyWord kw_210[3]  [static]

Initial value:

```
{
    "descriptors",15,0,3,0,0,0.,0.,0.,0,N_vae(dailbl,DAUIVar_negative_binomial)),
    "num_trials",13,0,2,2,0,0,0,0,N_vam(intDL,negBinomialUncNumTrials)),
    "prob_per_trial",14,0,1,1,0,0,0,0,0,N_vam(RealLd,negBinomialUncProbPerTrial))
```
### 7.1.3.245 KeyWord kw_211[10] [static]

Initial value:

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

### 7.1.3.246 KeyWord kw_212[2] [static]

Initial value:

```json
{
    "descriptors", 15, 0, 2, 0, 0, 0, 0, 0, 0, N_vae(dailbl, DAUIVar_poisson),
    "lambdas", 14, 0, 1, 1, 0, 0, 0, 0, 0, N_vam(RealLd, poissonUncLambdas)
}
```

### 7.1.3.247 KeyWord kw_213[8] [static]

Initial value:

```json
{
    "descriptors", 15, 0, 4, 0, 0, 0, 0, 0, 0, N_vae(caulbl, CAUVar_triangular),
    "lower_bounds", 14, 0, 2, 2, 0, 0, 0, 0, 0, N_vam(RealLb, triangularUncLowerBnds),
    "modes", 14, 0, 1, 1, 0, 0, 0, 0, 0, N_vam(RealLd, triangularUncModes),
    "tuv_descriptors", 7, 0, 4, 0, 0, 0, 0, 0, -3, N_vae(caulbl, CAUVar_triangular),
    "tuv_lower_bounds", 6, 0, 2, 2, 0, 0, 0, 0, -3, N_vam(RealLb, triangularUncLowerBnds),
    "tuv_modes", 6, 0, 1, 1, 0, 0, 0, 0, -3, N_vam(RealLd, triangularUncModes),
    "tuv_upper_bounds", 6, 0, 3, 3, 0, 0, 0, 0, 1, N_vam(RealUb, triangularUncUpperBnds),
    "upper_bounds", 14, 0, 3, 3, 0, 0, 0, 0, 0, N_vam(RealUb, triangularUncUpperBnds)
}
```

### 7.1.3.248 KeyWord kw_214[6] [static]

Initial value:

```json
{
    "descriptors", 15, 0, 3, 0, 0, 0, 0, 0, 0, N_vae(caulbl, CAUVar_uniform),
    "upper_bounds", 14, 0, 1, 1, 0, 0, 0, 0, 0, N_vam(RealUb, uniformUncUpperBnds),
    "uuv_descriptors", 7, 0, 3, 0, 0, 0, 0, 0, -3, N_vae(caulbl, CAUVar_uniform),
    "uuv_lower_bounds", 6, 0, 1, 1, 0, 0, 0, 0, -3, N_vam(RealLb, uniformUncLowerBnds),
    "uuv_upper_bounds", 6, 0, 2, 2, 0, 0, 0, 0, -3, N_vam(RealUb, uniformUncUpperBnds)
}
```
7.1 Dakota Namespace Reference

7.1.3.249 KeyWord kw_215[6] [static]

Initial value:

```{
    "alphas", 14, 0, 1, 0, 0, 0, 0, N_vam(RealLb, weibullUncAlphas),
    "betas", 14, 0, 2, 0, 0, 0, 0, N_vam(RealLb, weibullUncBetas),
    "descriptors", 15, 0, 3, 0, 0, 0, 0, 0, N_vae(caulbl, CAUVar_weibull),
    "wuv_alphas", 6, 0, 1, 0, 0, 0, -3, N_vam(RealLb, weibullUncAlphas),
    "wuv_betas", 6, 0, 2, 0, 0, 0, -3, N_vam(RealLb, weibullUncBetas),
    "wuv_descriptors", 7, 0, 3, 0, 0, 0, -3, N_vae(caulbl, CAUVar_weibull)
}
```

7.1.3.250 KeyWord kw_217[6] [static]

Initial value:

```{
    "interface", 0x308, 10, 5, 5, kw_9, 0, 0, 0, N_ifm3(start, 0, stop),
    "method", 0x308, 60, 2, 2, kw_135, 0, 0, 0, N_mdm3(start, 0, stop),
    "model", 8, 6, 3, 3, kw_158, 0, 0, 0, N_mom3(start, 0, stop),
    "responses", 0x308, 15, 6, 6, kw_175, 0, 0, 0, N_rem3(start, 0, stop),
    "strategy", 0x108, 9, 1, 1, kw_186, 0, 0, 0, NIDRProblemDescDB::strategy_start,
    "variables", 0x308, 29, 4, 4, kw_216, 0, 0, 0, 0, N_vam3(start, 0, stop)
}
```

7.1.3.251 Var_uinfo CAUVLbl[CAUVar_Nkinds] [static]

Initial value:

```{
    UncInfo(nuv_, Normal),
    UncInfo(lnuv_, Lognormal),
    UncInfo(uuv_, Uniform),
    UncInfo(luuv_, Loguniform),
    UncInfo(tuv_, Triangular),
    UncInfo(euv_, Exponential),
    UncInfo(beuv_, Beta),
    UncInfo(gauv_, Gamma),
    UncInfo(guvv_, Gumbel),
    UncInfo(luvv_, Frechet),
    UncInfo(wuv_, Weibull),
    UncInfo(hbuv_, HistogramBin)
}
```

7.1.3.252 Var_uinfo DAUIVLbl[DAUIVar_Nkinds] [static]

Initial value:

```{
    UncInfo(puv_, Poisson),
    UncInfo(puvv_, Poisson),
    UncInfo(puvvv_, Poisson)
}
```
UncInfo(biuv_, Binomial),
UncInfo(nbuv_, NegBinomial),
UncInfo(geuv_, Geometric),
UncInfo(hguv_, HyperGeom)
}

7.1.3.253 Var_uinfo DAURVLbl[DAURVar_Nkinds] [static]

Initial value:

{  UncInfo(hpuv_, HistogramPt)  }

7.1.3.254 Var_uinfo CEUVLbl[CEUVar_Nkinds] [static]

Initial value:

{  UncInfo(iuv_, Interval)  }

7.1.3.255 Var_uinfo DiscSetLbl[DiscSetVar_Nkinds] [static]

Initial value:

{  DiscSetInfo(ddsiv_, DesSetInt),
   DiscSetInfo(ddsrv_, DesSetReal),
   DiscSetInfo(dssiv_, StateSetInt),
   DiscSetInfo(dssrv_, StateSetReal)  }

7.1.3.256 VarLabelChk Vlch[ ] [static]

Initial value:

{  { AVI numContinuousDesVars, AVI continuousDesignLabels, "cdv_", "cdv_descriptors" },
   { AVI numDiscreteDesRangeVars, AVI discreteDesignRangeLabels, "ddriv_", "ddriv_descriptors" },
   { AVI numDiscreteDesSetIntVars, AVI discreteDesignSetIntLabels, "ddsiv_", "ddsiv_descriptors" },
   { AVI numDiscreteDesSetRealVars, AVI discreteDesignSetRealLabels, "dssrv_", "dssrv_descriptors" },
   { AVI numContinuousStateVars, AVI continuousStateLabels, "csv_", "csv_descriptors" },
   { AVI numDiscreteStateRangeVars, AVI discreteStateRangeLabels, "dssriv_", "dssriv_descriptors" },
   { AVI numDiscreteStateSetIntVars, AVI discreteStateSetIntLabels, "dssiv_", "dssiv_descriptors" },
   { AVI numDiscreteStateSetRealVars, AVI discreteStateSetRealLabels, "dssrv_", "dssrv_descriptors" },
   { AVI numContinuousDesVars, AVI continuousDesignScaleTypes, 0, "cdv_scale_types" }  }
}
7.1 Dakota Namespace Reference

7.1.3.257 VLstuff VLS[N_VLS] [static]

Initial value:

```c
{
    {CAUVar_Nkinds, 1, AVI CAUv, CAUVLbl,
        DVR continuousAleatoryUncLabels,
        DVR continuousAleatoryUncLowerBnds,
        DVR continuousAleatoryUncUpperBnds,
        DVR continuousAleatoryUncVars},
    {CEUVar_Nkinds, 1, AVI CEUv, CEUVLbl,
        DVR continuousEpistemicUncLabels,
        DVR continuousEpistemicUncLowerBnds,
        DVR continuousEpistemicUncUpperBnds,
        DVR continuousEpistemicUncVars},
    {DAUIVar_Nkinds, 0, AVI DAUIv, DAUILbl,
        DVR discreteIntAleatoryUncLabels,
        RDVR discreteIntAleatoryUncLowerBnds,
        RDVR discreteIntAleatoryUncUpperBnds,
        RDVR discreteIntAleatoryUncVars},
    {DAURVar_Nkinds, 1, AVI DAURv, DAURVlbl,
        DVR discreteRealAleatoryUncLabels,
        DVR discreteRealAleatoryUncLowerBnds,
        DVR discreteRealAleatoryUncUpperBnds,
        DVR discreteRealAleatoryUncVars}
}
```

7.1.3.258 Var_bgen var_mp_bgen[ ] [static]

Initial value:

```c
{
    Vchu0(gamma_uncertain, numGammaUncVars, Gamma),
    Vchu0(gumbel_uncertain, numGumbelUncVars, Gumbel),
    Vchu0(frechet_uncertain, numFrechetUncVars, Frechet),
    Vchu0(weibull_uncertain, numWeibullUncVars, Weibull),
    Vchu0(histogram_bin_uncertain, numHistogramBinUncVars, HistogramBin)
}
```

7.1.3.259 Var_bgen var_mp_bgen_audr[ ] [static]

Initial value:

```c
{
    Vchu0(histogram_point_uncertain, numHistogramPtUncVars, HistogramPt)
}
```

7.1.3.260 Var_bgen var_mp_bgen_audi[ ] [static]

Initial value:

```c
{
}
```
{  
    Vchu0(poisson_uncertain,numPoissonUncVars,Poisson),  
    Vchu0(binomial_uncertain,numBinomialUncVars,Binomial),  
    Vchu0(negative_binomial_uncertain,numNegBinomialUncVars,NegBinomial),  
    Vchu0(geometric_uncertain,numGeometricUncVars,Geometric),  
    Vchu0(hypergeometric_uncertain,numHyperGeomUncVars,HyperGeom)  
}

7.1.3.261  Var_bgen var_mp_bgen_eu[ ]  [static]

Initial value:

{  
    Vchu0(interval_uncertain,numIntervalUncVars,Interval)  
}

7.1.3.262  Var_bgen var_mp_bgen_dis[ ]  [static]

Initial value:

{  
    Vchu0(discrete_design_set_integer,numDiscreteDesSetIntVars,DDSI),  
    Vchu0(discrete_design_set_real,numDiscreteDesSetRealVars,DDSR),  
    Vchu0(discrete_state_set_integer,numDiscreteStateSetIntVars,DSSI),  
    Vchu0(discrete_state_set_real,numDiscreteStateSetRealVars,DSSR)  
}

7.1.3.263  VarBgen Bgen[ ]  [static]

Initial value:

{  
    BgenInit(var_mp_bgen_audr),  
    BgenInit(var_mp_bgen_audi),  
    BgenInit(var_mp_bgen_eu)  
}

7.1.3.264  Var_bchk var_mp_bndchk[ ]  [static]

Initial value:

{  
    Vchv(continuous_design,numContinuousDesVars,continuousDesign),  
    Vchu1(normal_uncertain,numNormalUncvars,normalUnc),  
    Vchu1(lognormal_uncertain,numLognormalUncVars,lognormalUnc),  
    Vchu(uniform_uncertain,numUniformUncVars,uniformUnc),  
    Vchu(loguniform_uncertain,numLoguniformUncVars,loguniformUnc),  
    Vchu(triangular_uncertain,numTriangularUncVars,triangularUnc),  
    Vchu0(exponential_uncertain,numExponentialUncVars,Exponential),  
    Vchu(beta_uncertain,numBetaUncVars,betaUnc),  
    Vchv(continuous_state,numContinuousStateVars,continuousState)  
}
7.1 Dakota Namespace Reference

7.1.3.265 Var_ibchk var_mp_ibndchk[ ] [static]

Initial value:

```c
{
    Vchi(discrete_design_range, numDiscreteDesRangeVars, discreteDesignRange),
    Vchi(discrete_state_range, numDiscreteStateRangeVars, discreteStateRange)
}
```
7.2 SIM Namespace Reference

plug facilities into DAKOTA.

Classes

- class ParallelDirectApplicInterface
  plug-ins using assign_rep().

- class SerialDirectApplicInterface
  plug-ins using assign_rep().

7.2.1 Detailed Description

plug facilities into DAKOTA.

A typical use of plug-ins with assign_rep() is to publish a simulation interface for use in library mode. See Interfacing with DAKOTA as a Library for more information.
Chapter 8

DAKOTA Class Documentation

8.1 ActiveSet Class Reference

active set request vector and the derivative variables vector.

Public Member Functions

- **ActiveSet ()**
  
  *default constructor*

- **ActiveSet (size_t num_fns, size_t num_deriv_vars)**
  
  *standard constructor*

- **ActiveSet (const ActiveSet &set)**
  
  *copy constructor*

- **~ActiveSet ()**
  
  *destructor*

- **ActiveSet & operator= (const ActiveSet &set)**
  
  *assignment operator*

- **void reshape (size_t num_fns, size_t num_deriv_vars)**
  
  *reshape requestVector and derivVarsVector*

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

- **void request_vector (const ShortArray &rv)**
  
  *set the request vector*
void request_values (const short rv_val)
set all request vector values

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

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

void derivative_vector (const UIntArray &dvv)
set the derivative variables vector from a UIntArray

void derivative_vector (UIntMultiArrayConstView dvv)
set the derivative variables vector from a UIntMultiArrayConstView

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

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

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

void write_annotated (std::ostream &s) const
write an active set object to an std::ostream in annotated format

void read (BiStream &s)
read an active set object from the binary restart stream

void write (BoStream &s) const
write an active set object to the binary restart stream

void read (MPIUnpackBuffer &s)
read an active set object from a packed MPI buffer

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

Private Attributes

ShortArray requestVector
the vector of response requests

 UIntArray derivVarsVector
the vector of variable ids used for computing derivatives
Friends

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

8.1.1 Detailed Description

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

8.1.2 Member Data Documentation

8.1.2.1 ShortArray requestVector [private]
the vector of response requests
It uses a 0 value for inactive functions and sums 1 (value), 2 (gradient), and 4 (Hessian) for active functions.

8.1.2.2 UIntArray derivVarsVector [private]
the vector of variable ids used for computing derivatives
These ids will generally identify either the active continuous variables or the inactive continuous variables.
The documentation for this class was generated from the following files:

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

processes for managing simulations.

Inheritance diagram for AnalysisCode::

```
AnalysisCode
    ForkAnalysisCode
    SysCallAnalysisCode
```

Public Member Functions

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

- void `write_parameters_files` (const Variables &vars, const ActiveSet &set, const int id)
  write parameters_file() in either standard or apprepro format

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

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

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

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

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

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

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

- void `suppress_output_flag` (const bool flag)
  set suppressOutputFlag

- bool `suppress_output_flag` () const
  return suppressOutputFlag
8.2 AnalysisCode Class Reference

- bool command_line_arguments () const
  
  return commandLineArgs

- bool multiple_parameters_filenames () const
  
  return multipleParamsFiles

- const char * workdir () const
  
  return Workdir if useWorkdir is true

- void file_cleanup () const
  
  remove temporary files if not fileSaveFlag

Protected Member Functions

- AnalysisCode (const ProblemDescDB &problem_db)
  
  constructor

- ~AnalysisCode ()
  
  destructor

Protected Attributes

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

- short outputLevel
  
  output verbosity level: {SILENT,QUIET,NORMAL,VERBOSE,DEBUG}_OUTPUT

- bool fileTagFlag
  
  flags tagging of parameter/results files

- bool fileSaveFlag
  
  flags retention of parameter/results files

- bool commandLineArgs
  
  the analysis drivers and input/output filters

- bool apreproFlag
  
  format for parameter files

- bool multipleParamsFiles
  
  analysis drivers
• **String iFilterName**
  
  the name of the input filter (input_filter user specification)

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

• **StringArray programNames**
  
  specification)

• **size_t numPrograms**
  
  the number of analysis code programs (length of programNames)

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

• **String paramsFileName**
  
  temp files)

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

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

• **String curWorkdir**
  
  working_directory when useWorkdir is true

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

• **bool useWorkdir**
  
  whether to use a new or specified work_directory

• **String workDir**
  
  its name, if specified...

• **bool dirTag**
  
  whether to tag the working_directory

• **bool dirSave**
  
  whether dir_save was specified

• **bool dirDel**
  
  whether to delete the directory when Dakota terminates
• String **templateDir**
  template directory (if specified)

• StringArray **templateFiles**
  template files (if specified)

• bool **templateCopy**
  whether to force a copy (versus link) every time

• bool **templateReplace**
  whether to replace existing files

• bool **haveTempdir**
  state variable for \texttt{working\_directory}

• bool **haveWorkdir**
  for \texttt{dirTag}, whether we have \texttt{workDir}

• String **dakDir**
  \texttt{Dakota} directory (if needed).

**Private Member Functions**

• void **write\_parameters\_file** (const Variables &\texttt{vars}, const ActiveSet &\texttt{set}, const StringArray &\texttt{an\_comps}, const String &\texttt{params\_fname})
  standard or \texttt{aprepro} format

**Private Attributes**

• ParallelLibrary & **parallelLib**
  reference to the \texttt{ParallelLibrary} object. Used in \texttt{define\_filenames()}.  

• String2DArray **analysisComponents**
  \texttt{(from the analysis\_components interface specification)}

**8.2.1 Detailed Description**

processes for managing simulations.

The **AnalysisCode** class hierarchy provides simulation spawning services for **ApplicationInterface** derived classes and alleviates these classes of some of the specifics of simulation code management. The hierarchy does not employ the letter-envelope technique since the **ApplicationInterface** derived classes instantiate the appropriate derived **AnalysisCode** class directly.
The documentation for this class was generated from the following files:

- AnalysisCode.H
- AnalysisCode.C
8.3 Analyzer Class Reference

hierarchy.

Inheritance diagram for Analyzer::

```
Analyzer
  |
  v
Iterator
  |
  v
NonD
  |
  v
NonDBayesCal
  |
  v
NonDExpansion
  |
  v
NonDIntegration
  |
  v
NonDInterval
  |
  v
NonDReliability
  |
  v
NonDSampling
  |
  v
PStudyDACE
  |
  v
DDACEDesignCompExp
  |
  v
FSUDesignCompExp
  |
  v
ParamStudy
  |
  v
PSUADesignCompExp
```

Public Member Functions

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

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

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

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

- `const Variables & variables_results () const`
  
  return a single final iterator solution (variables)

- `const Response & response_results () const`
  
  return a single final iterator solution (response)

- `void response_results_active_set (const ActiveSet &set)`
set the requested data for the final iterator response results

Protected Member Functions

- **Analyzer ()**
  
  *default constructor*

- **Analyzer (Model &model)**
  
  *standard constructor*

- **Analyzer (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor for instantiations "on the fly" with a Model*

- **Analyzer (NoDBBaseConstructor)**
  
  *alternate constructor for instantiations "on the fly" without a Model*

- **~Analyzer ()**
  
  *destructor*

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

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

- **void print_results (std::ostream &s)**
  
  *print the final iterator results*

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

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

- **void read_variables_responses (int num_evals)**
  
  *read num_evals variables/responses from file*

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

Protected Attributes

- **VariablesArray allVariables**
  
  *array of all variables evaluated*
• **ResponseArray allResponses**  
  array of all responses computed

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

• size_t numObjFns  
  number of objective functions

• size_t numLSqTerms  
  number of least squares terms

### Private Member Functions

• void update_best (const Variables &vars, const Response &response, const int eval_num)  
  compares current evaluation to best evaluation and updates best

### Private Attributes

• Variables bestVariables  
  best variables found during the study

• Response bestResponse  
  best response found during the study

• VariablesArray bestVariablesArray  
  collection of N best solution variables found during the study

• ResponseArray bestResponseArray  
  collection of N best solution responses found during the study

• size_t numSolnsTransferred  
  number of solutions to transfer in sequential hybrid study

• Real bestObjFn  
  best objective function found during the study

• Real bestConViol  
  precedence over objective function reduction.

• RealReal ParamRespMap bestVarsRespMap  
  map which stores best set of solutions
8.3.1 Detailed Description

hierarchy.

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

8.3.2 Member Function Documentation

8.3.2.1 void print_results (std::ostream & s) [protected, virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Iterator.

Reimplemented in PSTudyDACE, NonDBayesCal, NonDExpansion, NonDGlobalReliability, NonDIncrementalLHSampling, NonDInterval, NonDLHSSampling, NonDLocalReliability, and NonDPolynomialChaos.

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

into response sets (allResponses)

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

8.3.2.3 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.

8.3.2.4 void print_vbd (std::ostream & s, const RealVectorArray & S, const RealVectorArray & T) const [protected]

Printing of VBD results.

printing of variance based decomposition indices.

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

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

interfaces to simulation codes.

Inheritance diagram for ApplicationInterface::

```
Interface
  ApplicationInterface
    DirectApplicInterface
    ForkApplicInterface
    GridApplicInterface
    SysCallApplicInterface
    ParallelDirectApplicInterface
    SerialDirectApplicInterface
```

Public Member Functions

- **ApplicationInterface** (const ProblemDescDB &problem_db)
  
  *constructor*

- **~ApplicationInterface** ()
  
  *destructor*

Protected Member Functions

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

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

- void **free_communicators** ()
  
  *iterator and concurrent multiprocessor analyses within an evaluation.*

- void **init_serial** ()

- int **asynch_local_evaluation_concurrency** () const
  
  return asynchLocalEvalConcurrency

- String **interface_synchronization** () const
  
  return interfaceSynchronization

- void **map** (const Variables &vars, const ActiveSet &set, Response &response, const bool asynch_flag=false)
  
  *Protected due to Interface letter-envelope idiom.*

- void **manage_failure** (const Variables &vars, const ActiveSet &set, Response &response, int failed_eval_id)
manages a simulation failure using abort/retry/recover/continuation

- const IntResponseMap & synch ()
  beforeSynchCorePRPQueue and returns all jobs

- const IntResponseMap & synch_nowait ()
  beforeSynchCorePRPQueue and returns a partial set of completed jobs

- void serve_evaluations ()
  run on evaluation servers to serve the iterator master

- void stop_evaluation_servers ()
  used by the iterator master to terminate evaluation servers

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

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

- virtual void derived_synch (PRPQueue &prp_queue)
  classes. This version waits for at least one completion.

- virtual void derived_synch_nowait (PRPQueue &prp_queue)
  any completions if none are immediately available.

- void self_schedule_analyses ()
  evaluation using message passing

- void serve_analyses_synch ()
  analysis job at a time

- virtual int derived_synchronous_local_analysis (const int &analysis_id)
  ApplicationInterface::serve_analyses_synch().

Protected Attributes

- ParallelLibrary & parallelLib
  the concurrent evaluations and concurrent analyses parallelism levels

- bool suppressOutput
  flag for suppressing output on slave processors

- int evalCommSize
size of evalComm

- int evalCommRank
  processor rank within evalComm

- int evalServerId
  evaluation server identifier

- bool eaDedMasterFlag
  flag for dedicated master partitioning at ea level

- int analysisCommSize
  size of analysisComm

- int analysisCommRank
  processor rank within analysisComm

- int analysisServerId
  analysis server identifier

- int numAnalysisServers
  number of analysis servers

- bool multiProcAnalysisFlag
  flag for multiprocessor analysis partitions

- bool asynchLocalAnalysisFlag
  flag for asynchronous local parallelism of analyses

- int asynchLocalAnalysisConcurrency
  scheduling and specifies hybrid concurrency when message passing

- int numAnalysisDrivers
  (from the analysis_drivers interface specification)

- IntSet completionSet
  and derived_synch_nowait()

### Private Member Functions

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

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

• void asynchronous_local_evaluations (PRPQueue &prp_queue)  
  the local processor

• void asynchronous_local_evaluations_static (PRPQueue &prp_queue)  
  asynchLocalEvalConcurrency

• void synchronous_local_evaluations (PRPQueue &prp_queue)  
  the local processor

• void asynchronous_local_evaluations_nowait (PRPQueue &prp_queue)  
  static-scheduling cases)

• void serve_evaluations_synch ()  
  one synchronous evaluation at a time

• void serve_evaluations_asynch ()  
  multiple asynchronous evaluations

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

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

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

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

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

• void continuation (const Variables &target_vars, const ActiveSet &set, Response &response, const ParamResponsePair &source_pair, int failed_eval_id)  
  Invoked by manage_failure() for failAction == "continuation".

• void common_input_filtering (const Variables &vars)  
  common input filtering operations, e.g. mesh movement

• void common_output_filtering (Response &response)  
  common output filtering operations, e.g. data filtering
Private Attributes

- int worldSize
  size of MPI_COMM_WORLD

- int worldRank
  processor rank within MPI_COMM_WORLD

- int iteratorCommSize
  size of iteratorComm

- int iteratorCommRank
  processor rank within iteratorComm

- bool ieMessagePass
  flag for message passing at ie scheduling level

- int numEvalServers
  number of evaluation servers

- bool eaMessagePass
  flag for message passing at ea scheduling level

- int procsPerAnalysis
  processors per analysis servers

- int lenVarsMessage
  computed in Model::init_communicators()

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

- int lenResponseMessage
  computed in Model::init_communicators()

- int lenPRPairMessage
  computed in Model::init_communicators()

- String evalScheduling
  auto-configure logic in ParallelLibrary::resolve_inputs().

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

- int asynchLocalEvalConcurrency
  scheduling and specifies hybrid concurrency when message passing
- `bool asynchLocalEvalStatic`  
  with a static schedule (default false)

- `IntArray localServerJobMap`  
  asynchronous local static schedules

- `String interfaceSynchronization`  
  or asynchronous

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

- `bool asvControlFlag`  
  on each evaluation.

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

- `bool restartFileFlag`  
  insertions into write_restart).

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

- `String failAction`  
  retry, recover, or continuation

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

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

- `IntResponseMap historyDuplicateMap`  
  evaluations. Map key is fnEvalId, map value is corresponding response.

- `std::map< int, std::pair< PRPQueueHIter, Response > > beforeSynchDuplicateMap`  
  beforeSynchCorePRPQueue evaluations

- `PRPQueue beforeSynchCorePRPQueue`  
  that is later scheduled in synch() or synch_nowait().

- `PRPQueue beforeSynchAlgPRPQueue`  
  that is later evaluated in synch() or synch_nowait().
8.4 ApplicationInterface Class Reference

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

8.4.1 Detailed Description

interfaces to simulation codes.

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

8.4.2 Member Function Documentation

8.4.2.1 void init_serial () [inline, protected, virtual]

DataInterface.C defaults of 0 servers are needed to distinguish an explicit user request for 1 server (serialization of a parallelism level) from no user request (use parallel auto-config). This default causes problems when init_communicators() is not called for an interface object (e.g., static scheduling fails in DirectApplication::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.

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

Protected due to Interface letter-envelope idiom.

The function evaluator for application interfaces. Called from derived_compute_response() and derived_asynch_compute_response() in derived Model classes. If asynch_flag is not set, perform a blocking evaluation (using derived_map()). If asynch_flag is set, add the job to the beforeSynchCorePRPQueue queue for execution by one of the scheduler routines in synch() or synch_nowait(). Duplicate function evaluations are detected with duplication_detect().

Reimplemented from Interface.

8.4.2.3 const IntResponseMap & synch () [protected, virtual]

beforeSynchCorePRPQueue and returns all jobs

This function provides blocking synchronization for all cases of asynchronous evaluations, including the local asynchronous case (background system call, nonblocking fork, & multithreads), the message passing case, and the hybrid case. Called from derived_synchonize() in derived Model classes.

Reimplemented from Interface.
8.4.2.4  const IntResponseMap & synch_nowait()  [protected, virtual]

beforeSynchCorePRPQueue and returns a partial set 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.

8.4.2.5  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.

8.4.2.6  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.

8.4.2.7  void self_schedule_analyses()  [protected]

evaluation using message passing
This code is called from derived classes to provide the master portion of a master-slave algorithm for the dynamic self-scheduling of analyses among slave servers. It is patterned after self_schedule_evaluations(). It performs no analyses locally and matches either serve_analyses_synch() or serve_analyses_asynch() on the slave servers, depending on the value of asynchLocalAnalysisConcurrency. Self-scheduling approach assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalAnalysisConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within Parallel-Library.

8.4.2.8  void serve_analyses_synch()  [protected]

analysis job at a time
This code is called from derived classes to run synchronous analyses on slave processors. The slaves receive requests (blocking receive), do local derived_map_ac’s, and return codes. This is done continuously until a termination signal is received from the master. It is patterned after serve_evaluations_synch().
8.4.2.9  bool duplication_detect (const Variables & vars, Response & response, const bool asynch_flag) [private]

evaluation request has already been performed or queued

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

8.4.2.10  void self_schedule_evaluations () [private]

using message passing; executes on iteratorComm master

This code is called from synch() to provide the master portion of a master-slave algorithm for the dynamic self-scheduling of evaluations among slave servers. It performs no evaluations locally and matches either serve_evaluations_synch() or serve_evaluations_asynch() on the slave servers, depending on the value of asynchLocalEvalConcurrency. Self-scheduling approach assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalEvalConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

8.4.2.11  void static_schedule_evaluations () [private]

using message passing; executes on iteratorComm master

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synch() in order to assign a static schedule. It matches serve_evaluations_peer() for any other processors within the 1st evaluation partition and serve_evaluations_synch()/serve_evaluations_asynch() for all other evaluation partitions (depending on asynchLocalEvalConcurrency). It performs function evaluations locally for its portion of the static schedule using either asynchronous_local_evaluations() or synchronous_local_evaluations(). Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary. The iteratorCommRank 0 processor assigns the static schedule since it is the only processor with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

8.4.2.12  void asynchronous_local_evaluations (PRPQueue & prp_queue) [private]

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()).
8.4.2.13 void asynchronous_local_evaluations_static (PRPQueue & prp_queue) [private]

asyncLocalEvalConcurrency
Locally statically-scheduled counterpart to asynchronous_local_evaluations. This scheduling policy specifically ensures that a completed asynchronous evaluation eval_id is replaced with an equivalent one, modulo asyncLocalEvalConcurrency. Designed to help with parallel tiling. A disadvantage of this scheduling policy is that it could leave local asynchronous worker "servers" idle in parsing the prp_queue, e.g., when restarting and some evals are already complete. In fact, anytime this function is called with non-contiguous eval_id’s the full possible concurrency won’t be leveraged.

This is currently only supported when DAKOTA is running in serial. Supporting in the MPI static / async local hybrid mode would require MPI static schedule that is either fully round-robin or fully block scheduled, not the present hybrid. It is not clear how to support this in the MPI self scheduled / async local hybrid mode.

If local evaluation concurrency is unlimited, this function is not needed.

8.4.2.14 void synchronous_local_evaluations (PRPQueue & prp_queue) [private]

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.

8.4.2.15 void asynchronous_local_evaluations_nowait (PRPQueue & prp_queue) [private]

static-scheduling cases)
This function provides nonblocking synchronization for the local async case (background system call, non-blocking fork, or threads). It is called from synch_nowait() and passed the complete set of all asynchronous jobs (beforeSynchCorePRPQueue). It uses derived_map_asynch() to initiate asynchronous evaluations and derived_synch_nowait() to capture completed jobs in nonblocking mode. It mirrors a nonblocking message passing scheduler as much as possible (derived_synch_nowait() modeled after MPI_Testsome()). The result of this function is rawResponseMap, which uses eval_id as a key. It is assumed that the incoming prp_queue contains only active and new jobs - i.e., all completed jobs are cleared by synch_nowait().

Also supports asynchronous local evaluations with static scheduling. This scheduling policy specifically ensures that a completed asynchronous evaluation eval_id is replaced with an equivalent one, modulo asyncLocalEvalConcurrency. In the nowait case, this could render some servers idle if evaluations don’t come in eval_id order or some evaluations are cancelled by the caller in between calls. If this function is called with unlimited local eval concurrency, the static scheduling request is ignored.

8.4.2.16 void serve_evaluations_synch () [private]

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()`).

### 8.4.2.17 `void serve_evaluations_asynch()` [private]

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).

### 8.4.2.18 `void serve_evaluations_peer()` [private]

one synchronous evaluation at a time as part of the 1st peer

This code is invoked by `serve_evaluations()` to perform a synchronous evaluation in coordination with the iterator-CommRank 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`
8.5 Approximation Class Reference

Base class for the approximation class hierarchy.

Inheritance diagram for Approximation:

```
Approximation
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BasisPolyApproximation GaussProcApproximation SurfpackApproximation TANA3Approximation TaylorApproximation</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>InterpPolyApproximation OrthogPolyApproximation</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **Approximation ()**
  default constructor

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

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

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

- **virtual ~Approximation ()**
  destructor

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

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

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

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

- **virtual const Real & get_variance (const RealVector &x)**
  retrieve the variance of the predicted value for a given parameter vector

- **virtual const Real & get_diagnostic (const String &metric_type)**
8.5 Approximation Class Reference

retrieve the diagnostic metric for the diagnostic type specified

- virtual const RealVector & approximation_coefficients () const
  return the coefficient array computed by find_coefficients()

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

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

- virtual int min_coefficients () const
  build the derived class approximation type in numVars dimensions

- virtual int recommended_coefficients () const
  build the derived class approximation type in numVars dimensions

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

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

- virtual const bool diagnostics_available ()
  check if diagnostics are available for this approximation type

- int min_samples (bool constraint_flag) const
  type in numVars dimensions. Uses _coefficients() and num_constraints().

- int recommended_samples (bool constraint_flag) const
  in numVars dimensions (default same as min_samples)

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

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

- const SurrogateDataPoint & anchor_point () const
  return anchorPoint

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

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

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

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

- void append (const RealVector &c_vars, const Real &fn_val, const RealVector &fn_grad, const RealSymMatrix &fn_hess)
  appends one additional entry to currentPoints

- void append (const VariablesArray &vars_array, const ResponseArray &resp_array, const int &fn_index)
  appends multiple additional entries to currentPoints

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

- bool anchor () const
  queries the status of anchorPoint

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

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

- void draw_surface ()
  problems only

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

**Protected Member Functions**

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

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

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

- short outputLevel
  
  output verbosity level: [SILENT, QUIET, NORMAL, VERBOSE, DEBUG]_OUTPUT

- int numVars
  
  number of variables in the approximation

- String approxType
  
  approximation type identifier

- UShortArray approxOrder
  
  orthogonal polynomials, and Taylor series)

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

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

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

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

- Real approxDiagnostic
  
  value of the diagnostic returned by get_diagnostic()

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

- SurrogateDataPoint anchorPoint
  
  least squares regression).

Private Member Functions

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

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

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

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

- void add_anchor (const RealVector &x, const Real &fn_val, const RealVector &fn_grad, const RealSymMatrix &fn_hess)
  
  add a new data point by assigning to anchorPoint

Private Attributes

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

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

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

- int referenceCount
  
  number of objects sharing approxRep

8.5.1 Detailed Description

Base class for the approximation class hierarchy.

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

8.5.2 Constructor & Destructor Documentation

8.5.2.1 Approximation ()

default constructor
The default constructor is used in Array<Approximation> instantiations and by the alternate envelope constructor. approxRep is NULL in this case (problem_db is needed to build a meaningful Approximation object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

8.5.2.2 Approximation (ProblemDescDB & problem_db, const size_t & num_vars)

standard constructor for envelope

Envelope constructor only needs to extract enough data to properly execute get_approx, since Approximation(BaseConstructor, problem_db) builds the actual base class data for the derived approximations.

8.5.2.3 Approximation (const String & approx_type, const UShortArray & approx_order, const size_t & num_vars)

alternate constructor

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

8.5.2.4 Approximation (const Approximation & approx)

copy constructor

Copy constructor manages sharing of approxRep and incrementing of referenceCount.

8.5.2.5 Approximation () [virtual]

destructor

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

8.5.2.6 Approximation (BaseConstructor, const ProblemDescDB & problem_db, const size_t & num_vars) [protected]

derived class constructors - Coplien, p. 139)

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

8.5.3 Member Function Documentation

8.5.3.1 Approximation operator= (const Approximation & approx)

assignment operator

### 8.5.3.2 void clear_current ()  [inline, virtual]

clear current build data in preparation for next build

Redefined by TANA3Approximation to clear current data but preserve history.

Reimplemented in TANA3Approximation.

### 8.5.3.3 void clear_all ()  [inline]

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

Clears out any history (e.g., TANA3Approximation use for a different response function in NonDReliability).

### 8.5.3.4 Approximation ∗ get_approx (ProblemDescDB & problem_db, const size_t & num_vars)  [private]

approxRep to the appropriate derived type.

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

### 8.5.3.5 Approximation ∗ get_approx (const String & approx_type, const UShortArray & approx_order, const size_t & num_vars)  [private]

approxRep to the appropriate derived type.

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

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

- DakotaApproximation.H
- DakotaApproximation.C
### 8.6 ApproximationInterface Class Reference

Approximations to simulation-based results.

Inheritance diagram for ApproximationInterface::

```
+ Interface
  - ApproximationInterface
```

#### Public Member Functions

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

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

- **~ApproximationInterface** ()
  *destructor*

#### Protected Member Functions

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

- **int minimum_samples** (bool constraint_flag) const
  *functionSurfaces*

- **int recommended_samples** (bool constraint_flag) const
  *functionSurfaces*

- **void approximation_function_indices** (const IntSet &approx_fn_indices)
  *set the (currently active) approximation function index set*

- **void update_approximation** (const Variables &vars, const Response &response)
- **void update_approximation** (const VariablesArray &vars_array, const ResponseArray &resp_array)
- **void append_approximation** (const Variables &vars, const Response &response)
- **void append_approximation** (const VariablesArray &vars_array, const ResponseArray &resp_array)
• void **build_approximation** (const BoolDeque &rebuild_deque, const RealVector &lower_bnds, const RealVector &upper_bnds)

  clears current data from an approximation interface

• void **clear_all** ()

  clears all data from an approximation interface

• bool **anchor** () const

  queries the presence of an anchorPoint within an approximation interface

• const **SurrogateDataPoint & anchor_point** () const

  returns the anchorPoint used within an approximation interface

• **Array< Approximation > & approximations** ()

  retrieve the Approximations within an ApproximationInterface

• const **RealVectorArray & approximation_coefficients** ()

  within an ApproximationInterface

• void **approximation_coefficients** (const RealVectorArray &approx_coeffs)

  within an ApproximationInterface

• void **print_coefficients** (std::ostream &s, size_t index) const

  Approximation instance within an ApproximationInterface.

• const **RealVector & approximation_variances** (const RealVector &c_vars)

  within an ApproximationInterface

• const **List< SurrogateDataPoint > & approximation_data** (size_t index)

  within an ApproximationInterface

• const **IntResponseMap & synch** ()

  recovers data from a series of asynchronous evaluations (blocking)

• const **IntResponseMap & synch_nowait** ()

  recovers data from a series of asynchronous evaluations (nonblocking)

**Private Attributes**

• IntSet **approxFnIndices**

  response function subset that is approximated

• **Array< Approximation > functionSurfaces**

  list of approximations, one per response function
• RealVectorArray functionSurfaceCoeffs
  response function
• RealVector functionSurfaceVariances
  vector of approximation variances, one value per response function
• List< SurrogateDataPoint > functionSurfaceDataPoints
  for a particular response function
• bool graph3DFlag
  controls 3D graphics of approximation surfaces
• StringArray diag_list
  List of diagnostic metrics.
• Variables actualModelVars
  among differing variable views
• IntResponseMap beforeSynchResponseMap
  but asynchronous virtual functions are supported through bookkeeping).

8.6.1 Detailed Description

approximations to simulation-based results.

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

8.6.2 Member Function Documentation

8.6.2.1 void update_approximation (const Variables & vars, const Response & response) [protected, virtual]

This function populates/replaces each Approximation::anchorPoint with the incoming variables/response data point.
Reimplemented from Interface.

8.6.2.2 void update_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array) [protected, virtual]

This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.
Reimplemented from Interface.
8.6.2.3  void append_approximation (const Variables & vars, const Response & response)  
[protected, virtual]

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

8.6.2.4  void append_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array)  [protected, virtual]

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

8.6.2.5  void build_approximation (const BoolDeque & rebuild_deque, const RealVector & lower_bnds,  
const RealVector & upper_bnds)  [protected, virtual]

This function finds the coefficients for each Approximation based on the data passed through update_approximation() calls. The bounds are used only for graphics visualization.
Reimplemented from Interface.

8.6.3  Member Data Documentation

8.6.3.1  Array<Approximation> functionSurfaces  [private]

list of approximations, one per response function

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

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

- ApproximationInterface.H
- ApproximationInterface.C
8.7 APPSEvalMgr Class Reference

Evaluation manager class for APPSPACK.

Public Member Functions

- **APPSEvalMgr (Model &model)**
  Evaluation manager class for APPSPACK.

- **~APPSEvalMgr ()**
  destructor

- **bool isWaiting () const**
  tells APPS whether or not there is a processor available to perform a function evaluation

- **bool spawn (const APPSPACK::Vector &x_in, int tag_in)**
  performs a function evaluation at APPS-provided x_in

- **int recv (int &tag_out, APPSPACK::Vector &f_out, string &msg_out)**
  returns a function value to APPS

- **void print () const**
  currently does nothing but is needed to complete the interface

- **void set_asynch_flag (const bool dakotaAsynchFlag)**
  publishes whether or not to do asynchronous evaluations

- **void set_blocking_synch (const bool blockingSynchFlag)**
  publishes whether or not APPS is operating synchronously

- **void set_total_workers (const int numDakotaWorkers)**
  publishes the number of processors available for function evaluations

- **void set_constraint_map (std::vector<int> constraintMapIndices, std::vector<double> constraintMapMultipliers, std::vector<double> constraintMapOffsets)**
  publishes constraint transformation

Private Attributes

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

- **bool modelAsynchFlag**
  flag for asynchronous function evaluations
8.7.1 Detailed Description

Evaluation manager class for APPSPACK.

The APPSEvalMgr class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 APPSEvalMgr (Model & model)

Evaluation manager class for APPSPACK.
The `APPSEvalMgr` class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.

### 8.7.3 Member Function Documentation

#### 8.7.3.1 bool isWaiting () const

tells APPS whether or not there is a processor available to perform a function evaluation

Check to see if all processors available for function evaluations are being used. If not, tell APPS that one is available.

#### 8.7.3.2 bool spawn (const APPSPACK::Vector & apps_xtrial, int apps_tag)

performs a function evaluation at APPS-provided x_in

Convert APPSPACK vector of variables to DAKOTA vector of variables and perform function evaluation asynchronously or not as specified in the DAKOTA input deck. If evaluation is asynchronous, map the dakota id to the APPS tag. If evaluation is synchronous, map the responses to the APPS tag.

#### 8.7.3.3 int recv (int & apps_tag, APPSPACK::Vector & apps_f, string & apps_msg)

returns a function value to APPS

Retrieve a set of response values, convert to APPS data structures, and return them to APPS. APPS tags are tied to corresponding responses using the appropriate (i.e., asynchronous or synchronous) map.

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

- APPSEvalMgr.H
- APPSEvalMgr.C
8.8 APPSOptimizer Class Reference

Wrapper class for APPSPACK.

Inheritance diagram for APPSOptimizer::

```
  Iterator
  Minimizer
  Optimizer
  APPSOptimizer
```

Public Member Functions

- `APPSOptimizer (Model &model)`
  Wrapper class for APPSPACK.

- `APPSOptimizer (NoDBBaseConstructor, Model &model)`
  alternate constructor for on-the-fly instantiations

- `~APPSOptimizer ()`
  destructor

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

Protected Member Functions

- `void set_apps_parameters ()`
  sets options for specific methods based on user specifications

- `void initialize_variables_and_constraints ()`
  initializes problem variables and constraints

Protected Attributes

- `APPS::Parameter::List params`
  Pointer to APPS parameter list.
8.8 APPSOptimizer Class Reference

- **APPSOptimizerr * evalMgr**
  
  Pointer to the APPSApplication object.

- **std::vector< int > constraintMapIndices**
  
  map from Dakota constraint number to APPS constraint number

- **std::vector< double > constraintMapMultipliers**
  
  multipliers for constraint transformations

- **std::vector< double > constraintMapOffsets**
  
  offsets for constraint transformations

### 8.8.1 Detailed Description

Wrapper class for APPSPACK.

The `APPSOptimizer` class provides a wrapper for APPSPACK, a Sandia-developed C++ library for generalized pattern search. APPSPACK defaults to a coordinate pattern search but also allows for augmented search patterns. It can solve problems with bounds, linear constraints, and general nonlinear constraints. `APPSOptimizer` uses an `APPSEvalMgr` object to manage the function evaluations.

The user input mappings are as follows: `output max_function_evaluations, constraint_tol initial_delta, contraction_factor, threshold_delta, solution_target, synchronization, merit_function, constraint_penalty, and smoothing_factor` are mapped into APPS’s "Debug", "Maximum Evaluations", "Bounds Tolerance"/"Machine Epsilon"/"Constraint Tolerance", "Initial Step", "Contraction Factor", "Step Tolerance", "Function Tolerance", "Synchronous", "Method", "Initial Penalty Value", and "Initial Smoothing Value" data attributes. Refer to the APPS web site ([http://software.sandia.gov/appspack](http://software.sandia.gov/appspack)) for additional information on APPS objects and controls.

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 APPSOptimizer (Model & model)

Wrapper class for APPSPACK.

The `APPSOptimizer` class provides a wrapper for APPSPACK, a Sandia-developed C++ library for generalized pattern search. APPSPACK defaults to a coordinate pattern search but also allows for augmented search patterns. It can solve problems with bounds, linear constraints, and general nonlinear constraints. `APPSOptimizer` uses an `APPSEvalMgr` object to manage the function evaluations.

The user input mappings are as follows: `output max_function_evaluations, constraint_tol initial_delta, contraction_factor, threshold_delta, solution_target, synchronization, merit_function, constraint_penalty, and smoothing_factor` are mapped into APPS’s "Debug", "Maximum Evaluations", "Bounds Tolerance"/"Machine Epsilon"/"Constraint Tolerance", "Initial Step", "Contraction Factor", "Step Tolerance", "Function Tolerance", "Synchronous", "Method", "Initial Penalty Value", and "Initial Smoothing Value" data attributes. Refer to the APPS web site ([http://software.sandia.gov/appspack](http://software.sandia.gov/appspack)) for additional information on APPS objects and controls.
8.8.3 Member Function Documentation

8.8.3.1 void find_optimum () [virtual]

Performs the iterations to determine the optimal solution.
find_optimum redefines the Optimizer virtual function to perform the optimization using APPS. It first sets up the problem data, then executes minimize() on the APPS optimizer, and finally catalogues the results.
Implements Optimizer.

8.8.3.2 void set_apps_parameters () [protected]

sets options for specific methods based on user specifications
Set all of the APPS algorithmic parameters as specified in the DAKOTA input deck. This is called at construction time.

8.8.3.3 void initialize_variables_and_constraints () [protected]

initializes problem variables and constraints
Set the variables and constraints as specified in the DAKOTA input deck. This is done at run time.
The documentation for this class was generated from the following files:

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

Template class for the Dakota bookkeeping array.

Public Member Functions

- **Array ()**
  Default constructor.

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

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

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

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

- **~Array ()**
  Destructor.

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

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

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

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

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

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

- **T & 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 i-th value of the array.

- void read (std::istream &s)
  
  Reads an Array from an std::istream.

- void write (std::ostream &s) const
  
  Writes an Array to an output stream.

- void write (std::ostream &s, const Array &label_array) const
  
  Writes an Array and associated label array to an output stream.

- void write_aprepro (std::ostream &s, const Array &label_array) const
  
  Writes an Array and associated label array to an output stream in aprepro format.

- void write.annotated (std::ostream &s, bool write.len) const
  
  Writes an Array to an output stream in annotated format.

- void read (BiStream &s)
  
  Reads an Array from a binary input stream.

- void write (BoStream &s) const
  
  Writes an Array to a binary output stream.

- void read (MPIUnpackBuffer &s)
  
  Reads an Array from a buffer after an MPI receive.

- void write (MPIPackBuffer &s) const
  
  Writes an Array to a buffer prior to an MPI send.

- size_t length () const
  
  Returns size of array.

- void reshape (size_t sz)
  
  Resizes array to size sz.

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

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

- const T * data () const
  
  Returns pointer T* to continuous data.
8.9 Array Class Template Reference

8.9.1 Detailed Description

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

Template class for the Dakota bookkeeping array.
An array class template that provides additional functionality that is specific to Dakota’s needs. The Array class adds additional functionality needed by Dakota to the inherited base (i.e. std::vector) class.

8.9.2 Constructor & Destructor Documentation

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

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

8.9.3 Member Function Documentation

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

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

8.9.3.2 operator T*() const [inline]

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

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

Index operator, returns the ith value of the array.
Index operator; calls the STL method at() which is bounds checked, but only when ‘–enable-debug’ is specified during configuration.

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

Index operator const, returns the ith value of the array.
A const version of the index operator; calls the STL method at() which is bounds checked if building a debug executable.
8.9.3.5  \texttt{const T \* data () const}  \texttt{[inline]}

Returns pointer T\* to continuous data.

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

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

- DakotaArray.H
8.10 BaseConstructor Struct Reference

Dummy struct for overloading letter-envelope constructors.

Public Member Functions

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

8.10.1 Detailed Description

Dummy struct for overloading letter-envelope constructors.

BaseConstructor is used to overload the constructor for the base class portion of letter objects. It avoids infinite recursion (Coplien p.139) in the letter-envelope idiom by preventing the letter from instantiating another envelope. Putting this struct here avoids circular dependencies.

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

• global defs.h
8.11 BasisPolyApproximation Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for BasisPolyApproximation:

```
Approximation
   BasisPolyApproximation
   InterpPolyApproximation
   OrthogPolyApproximation
```

**Public Member Functions**

- **BasisPolyApproximation ()**
  
  *default constructor*

- **BasisPolyApproximation (const ProblemDescDB &problem_db, const size_t &num_acv)**
  
  *standard constructor*

- **~BasisPolyApproximation ()**
  
  *destructor*

- virtual void allocate_arrays ()
  
  *size Sobol arrays*

- virtual void compute_global_sensitivity ()=0
  
  *Performs global sensitivity analysis using Sobol' Indices.*

- virtual const Real & get_mean ()=0
  
  *return the mean of the expansion, treating all variables as random*

- virtual const Real & get_mean (const RealVector &x)=0
  
  *treat a subset of the variables as random*

- virtual RealVector get_mean_gradient ()=0
  
  *vector, treating all variables as random*

- virtual const RealVector & get_mean_gradient (const RealVector &x, const UIntArray &dv)=0
  
  *and given DVV, treating a subset of the variables as random*

- virtual const Real & get_variance ()=0
  
  *return the variance of the expansion, treating all variables as random*
virtual const Real & get_variance (const RealVector &x)=0
   treating a subset of the variables as random

virtual const RealVector & get_variance_gradient ()=0
   vector, treating all variables as random

virtual const RealVector & get_variance_gradient (const RealVector &x, const UIntArray &dvv)=0
   vector and given DVV, treating a subset of the variables as random

virtual const Real & get_covariance (const RealVector &exp_coeffs_2)=0
   return the variance of the expansion, treating all variables as random

void solution_approach (short soln_approach)
   set expCoeffsSolnApproach

short solution_approach () const
   get expCoeffsSolnApproach

void expansion_coefficient_flag (bool coeff_flag)
   set expansionCoeffFlag

bool expansion_coefficient_flag () const
   get expansionCoeffFlag

void expansion_gradient_flag (bool grad_flag)
   set expansionGradFlag

bool expansion_gradient_flag () const
   get expansionGradFlag

const RealVector & sobol_indices () const
   return sobolIndices

const RealVector & total_sobol_indices () const
   return totalSobolIndices

void integration_iterator (const Iterator &iterator)
   set integrationRep

void random_variables_key (const BoolDeque &random_vars_key)
   set randomVarsKey

size_t tensor_product_terms (const UShortArray &order, bool exp_order_offset=false)
   expansion orders (offset = true)
Static Public Member Functions

- static size_t total_order_terms (const UShortArray &upper_bound, short lower_bound_offset=-1)
  
  with the provided (anisotropic) upper_bound array specification

- static void increment_indices (UShortArray &indices, const UShortArray &limits, bool include_limit_equality)
  
  utility function for incrementing a set of multidimensional indices

- static void increment_terms (UShortArray &terms, size_t &last_index, size_t &prev_index, const size_t &term_limit, bool &order_complete)
  
  utility function for incrementing a set of multidimensional terms

Protected Member Functions

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

- const RealVector & approximation_coefficients () const
  
  return the coefficient array computed by find_coefficients()

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

- void tensor_product_multi_index (const UShortArray &order, UShort2DArray &multi_index, bool exp_order_offset=false)
  
  initialize multiIndex using a tensor-product expansion

- void total_order_multi_index (const UShortArray &upper_bound, UShort2DArray &multi_index, short lower_bound_offset=-1)
  
  upper_bound array specification

- void total_order_multi_index (unsigned short upper_bound, const RealVector &anisotropic_wts, UShort2DArray &multi_index, RealArray &coeffs)
  upper_bound specification

Protected Attributes

- short expCoeffsSolnApproach
  
  QUADRATURE, SPARSE_GRID, REGRESSION, or SAMPLING.

- bool expansionCoeffFlag
  
  flag for calculation of expansionCoeffs from response values
8.11 BasisPolyApproximation Class Reference

- **bool expansionGradFlag**
  flag for calculation of expansionCoeffGrads from response gradients

- **NonDIntegration * integrationRep**
  weight products

- **BoolDeque randomVarsKey**
  the active variables (used in all_variables mode)

- **SizetList randomIndices**
  variables (used in all_variables mode; defined from randomVarsKey)

- **SizetList nonRandomIndices**
  active variables (used in all_variables mode; defined from randomVarsKey)

- **Real expansionMean**
  expected value of the expansion

- **RealVector expansionMeanGrad**
  gradient of the expected value of the expansion

- **Real expansionVariance**
  variance of the expansion

- **RealVector expansionVarianceGrad**
  gradient of the variance of the expansion

- **RealVector expansionCoeffs**
  the coefficients of the expansion

- **RealMatrix expansionCoeffGrads**
  the gradients of the expansion coefficients

- **RealVector sobolIndices**
  global sensitivities as given by Sobol'

- **RealVector totalSobolIndices**
  total global sensitivities as given by Sobol'

8.11.1 Detailed Description

Derived approximation class for global basis polynomials.

The **BasisPolyApproximation** class provides a global approximation based on basis polynomials. This includes orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic collocation.
8.11.2 Member Function Documentation

8.11.2.1 size_t total_order_terms (const UShortArray & upper_bound, short lower_bound_offset = -1) [static]

with the provided (anisotropic) upper_bound array specification

Return the number of terms in a total-order expansion. For anisotropic expansion order, no simple expression is currently available and the number of expansion terms is computed using the multiIndex recursion.

8.11.2.2 size_t tensor_product_terms (const UShortArray & order, bool exp_order_offset = false)

expansion orders (offset = true)

Return the number of terms in a tensor-product expansion. For isotropic and anisotropic expansion orders, calculation of the number of expansion terms is straightforward: \( \text{Prod}(p_i + 1) \).

8.11.3 Member Data Documentation

8.11.3.1 RealMatrix expansionCoeffGrads [protected]

the gradients of the expansion coefficients

may be interpreted as either the gradients of the expansion coefficients or the coefficients of expansions for the response gradients. This array is used when sensitivities of moments are needed with respect to variables that do not appear in the expansion (e.g., with respect to design variables for an expansion only over the random variables).

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

- BasisPolyApproximation.H
- BasisPolyApproximation.C
8.12 BasisPolynomial Class Reference

Base class for the basis polynomial class hierarchy.

Inheritance diagram for BasisPolynomial::

```
BasisPolynomial
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LagrangeInterpPolynomial</td>
</tr>
<tr>
<td>OrthogonalPolynomial</td>
</tr>
<tr>
<td>GenLaguerreOrthogPolynomial</td>
</tr>
<tr>
<td>HermiteOrthogPolynomial</td>
</tr>
<tr>
<td>JacobiOrthogPolynomial</td>
</tr>
<tr>
<td>LaguerreOrthogPolynomial</td>
</tr>
<tr>
<td>LegendreOrthogPolynomial</td>
</tr>
<tr>
<td>NumericGenOrthogPolynomial</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **BasisPolynomial ()**
  
  default constructor

- **BasisPolynomial (short poly_type)**
  
  alternate constructor

- **BasisPolynomial (const BasisPolynomial &polynomial)**
  
  copy constructor

- **virtual ~BasisPolynomial ()**
  
  destructor

- **BasisPolynomial operator= (const BasisPolynomial &polynomial)**
  
  assignment operator

- **virtual const Real & get_value (const Real &x, unsigned short n)**
  
  retrieve the basis polynomial value for a given parameter value

- **virtual const Real & get_gradient (const Real &x, unsigned short n)**
  
  retrieve the basis polynomial gradient for a given parameter value

- **virtual const Real & norm_squared (unsigned short n)**
  
  inner product $<\text{Poly}_n, \text{Poly}_n> = ||\text{Poly}_n||^2$

- **virtual const RealArray & gauss_points (unsigned short n)**
  
  return the gaussPoints corresponding to orthogonal polynomial order n.

- **virtual const RealArray & gauss_weights (unsigned short n)**
  
  return the gaussWeights corresponding to orthogonal polynomial order n.

- **virtual void reset_gauss ()**
  
  destroy history of Gauss pts/wts (due to change in alpha/beta stats)
• virtual const Real & point_factor ()
  (calculate and) return ptFactor

• virtual const Real & weight_factor ()
  (calculate and) return wtFactor

• virtual void alpha_polynomial (const Real &alpha)
  set \( \{ \text{Jacobi}, \text{GenLaguerre}\} \)OrthogPolynomial::alphaPoly

• virtual void beta_polynomial (const Real &beta)
  set JacobiOrthogPolynomial::betaPoly

• virtual void alpha_stat (const Real &alpha)
  GenLaguerreOrthogPolynomial::alphaPoly from statistical defn of alpha.

• virtual void beta_stat (const Real &beta)
  set JacobiOrthogPolynomial::alphaPoly from statistical defn of beta

• virtual void interpolation_points (const RealArray &interpolation_pts)
  set LagrangeInterpPolynomial::interpolationPts

• BasisPolynomial * polynomial_rep () const
  that are not mapped to the top BasisPolynomial level

• bool is_null () const
  function to check polyRep (does this handle contain a body)

### Static Public Member Functions

• static Real factorial (unsigned short n)
  compute \( n! \)

• static Real factorial_ratio (unsigned short num, unsigned short den)
  compute \( \text{num}/\text{den}! \)

• static Real n_choose_k (unsigned short n, unsigned short k)
  compute \( n!/(k!(n-k)!) \)

• static Real pochhammer (const Real &m, unsigned short n)
  compute the Pochhammer symbol \( (m)_n = m*(m+1)...*(m+n-1) \)
8.12 BasisPolynomial Class Reference

Protected Member Functions

- BasisPolynomial (BaseConstructor)
  derived class constructors - Coplien, p. 139)

Protected Attributes

- Real basisPolyValue
  value of the 1-D basis polynomial; returned by get_value()

- Real basisPolyGradient
  one parameter; returned by get_gradient()

- Real wtFactor
  weight discrepancy factor between Abramowitz-Stegun and PDF orthogonality

- Real ptFactor
  point discrepancy factor between Abramowitz-Stegun and PDF orthogonality

Private Member Functions

- BasisPolynomial * get_polynomial (short poly_type)
  appropriate derived type.

Private Attributes

- BasisPolynomial * polyRep
  pointer to the letter (initialized only for the envelope)

- int referenceCount
  number of objects sharing polyRep

8.12.1 Detailed Description

Base class for the basis polynomial class hierarchy.

The BasisPolynomial class is the base class for the univariate basis polynomial class hierarchy in DAKOTA. One instance of an BasisPolynomial is created for each variable within a multidimensional polynomial basis function (a vector of BasisPolynomials is contained in BasisPolyApproximation, which may be mixed and matched in, e.g., the Wiener-Askey scheme for polynomial chaos). For memory efficiency and enhanced polymorphism, the basis polynomial hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (BasisPolynomial) serves as the envelope and one of the derived classes (selected in BasisPolynomial::get_polynomial()) serves as the letter.
8.12.2 Constructor & Destructor Documentation

8.12.2.1 **BasisPolynomial ()**

default constructor

The default constructor is used in Array< BasisPolynomial > instantiations and by the alternate envelope constructor. polyRep is NULL in this case (problem_db is needed to build a meaningful instance). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

8.12.2.2 **BasisPolynomial (short poly_type)**

alternate constructor

Envelope constructor which does not require access to problem_db. This constructor executes get_polynomial(type), which invokes the default constructor of the derived letter class, which in turn invokes the BaseConstructor of the base class.

8.12.2.3 **BasisPolynomial (const BasisPolynomial & polynomial)**

copy constructor

Copy constructor manages sharing of polyRep and incrementing of referenceCount.

8.12.2.4 **~BasisPolynomial ()** [virtual]

destructor

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

8.12.2.5 **BasisPolynomial (BaseConstructor)** [protected]

derived class constructors - Coplien, p. 139)

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

8.12.3 Member Function Documentation

8.12.3.1 **BasisPolynomial operator= (const BasisPolynomial & polynomial)**

assignment operator

8.12 BasisPolynomial Class Reference

8.12.3.2 const Real & get_value (const Real &, unsigned short n) [virtual]

retrieve the basis polynomial value for a given parameter value

For orthogonal polynomials, n specifies the order of the polynomial, whereas for interpolation polynomials, it identifies the interpolant for the n-th point.


8.12.3.3 const Real & get_gradient (const Real &, unsigned short n) [virtual]

retrieve the basis polynomial gradient for a given parameter value

For orthogonal polynomials, n specifies the order of the polynomial, whereas for interpolation polynomials, it identifies the interpolant for the n-th point.


8.12.3.4 const Real & norm_squared (unsigned short n) [virtual]

inner product $<\text{Poly}_n, \text{Poly}_n> = ||\text{Poly}_n||^2$

This is defined only for orthogonal polynomials.


8.12.3.5 const RealArray & gauss_points (unsigned short n) [virtual]

return the gaussPoints corresponding to orthogonal polynomial order n.

This is defined only for orthogonal polynomials.


8.12.3.6 const RealArray & gauss_weights (unsigned short n) [virtual]

return the gaussWeights corresponding to orthogonal polynomial order n.

This is defined only for orthogonal polynomials.

8.12.3.7  void reset_gauss ()  [virtual]

destroy history of Gauss pts/wts (due to change in alpha/beta stats)
This is defined only for orthogonal polynomials.
Reimplemented in OrthogonalPolynomial.

8.12.3.8  void alpha_polynomial (const Real & alpha)  [virtual]

set {Jacobi,GenLaguerre}OrthogPolynomial::alphaPoly
This is defined only for parameterized orthogonal polynomials.
Reimplemented in GenLaguerreOrthogPolynomial, and JacobiOrthogPolynomial.

8.12.3.9  void beta_polynomial (const Real & beta)  [virtual]

set JacobiOrthogPolynomial::betaPoly
This is defined only for parameterized orthogonal polynomials.
Reimplemented in JacobiOrthogPolynomial.

8.12.3.10  void alpha_stat (const Real & alpha)  [virtual]

GenLaguerreOrthogPolynomial::alphaPoly from statistical defn of alpha.
This is defined only for parameterized orthogonal polynomials.
Reimplemented in GenLaguerreOrthogPolynomial, and JacobiOrthogPolynomial.

8.12.3.11  void beta_stat (const Real & beta)  [virtual]

set JacobiOrthogPolynomial::alphaPoly from statistical defn of beta
This is defined only for parameterized orthogonal polynomials.
Reimplemented in JacobiOrthogPolynomial.

8.12.3.12  void interpolation_points (const RealArray & interpolation_pts)  [virtual]

set LagrangeInterpPolynomial::interpolationPts
This is defined only for interpolation polynomials.
Reimplemented in LagrangeInterpPolynomial.

8.12.3.13  Real factorial (unsigned short n)  [inline, static]

compute n!
This implementation is unprotected from overflow, but this should be fine for the polynomial orders that we would expect to encounter. Whenever possible, orthogonal polynomial implementations should use `factorial_ratio()` or `n_choose_k()` instead of `factorial()` to avoid size_t overflow.

### 8.12.3.14 Real factorial_ratio (unsigned short num, unsigned short den) [inline, static]

compute num!/den!

This implementation sequences products in order to minimize the chances of overflow, and its use should be preferred to `factorial()` whenever possible.

### 8.12.3.15 Real n_choose_k (unsigned short n, unsigned short k) [inline, static]

compute n!/(k!(n-k)!)  

This implementation sequences products in order to minimize the chances of overflow, and its use should be preferred to `factorial()` whenever possible.

### 8.12.3.16 Real pochhammer (const Real & m, unsigned short n) [inline, static]

compute the Pochhammer symbol $(m)_n = m*(m+1)...*(m+n-1)$  

This is the rising/upper factorial formulation of the Pochhammer symbol $(m)_n$.

### 8.12.3.17 BasisPolynomial * get_polynomial (short poly_type) [private]

appropriate derived type.

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

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

- BasisPolynomial.H
- BasisPolynomial.C
8.13 BiStream Class Reference

data types

Public Member Functions

- BiStream ()
  Default constructor, need to open.

- BiStream (const char *s)
  Constructor takes name of input file.

- BiStream (const char *s, std::ios_base::openmode mode)
  Constructor takes name of input file, mode.

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

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

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

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

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

- BiStream & operator>>(int &i)
  Input operator, reads int* from binary stream BiStream.

- BiStream & operator>>(long &l)
  Input operator, reads long from binary stream BiStream.

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

- BiStream & operator>>(bool &b)
  Input operator, reads bool from binary stream BiStream.

- BiStream & operator>>(double &d)
  Input operator, reads double from binary stream BiStream.
8.13 BiStream Class Reference

- **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.*

### 8.13.1 Detailed Description

data types

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

### 8.13.2 Constructor & Destructor Documentation

#### 8.13.2.1 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.
8.13.2.2  **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.

8.13.2.3  **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.

8.13.2.4  **~BiStream ()**

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

8.13.3  **Member Function Documentation**

8.13.3.1  **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.

8.13.3.2  **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
8.14 BoStream Class Reference

data types

Public Member Functions

- **BoStream ()**
  
  *Default constructor, need to open.*

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

- **BoStream (const char *s, std::ios_base::openmode mode)**
  
  *Constructor takes name of input file, mode.*

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

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

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

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

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

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

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

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

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

- **BoStream & operator<<(const double &d)**
  
  *Output operator, writes double to binary stream BoStream.*
• **BoStream** & operator\(<<\) (const float &f)  
  *Output operator, writes float to binary stream BoStream.*

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

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

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

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

### Private Attributes

• XDR xdrOutBuf  
  *XDR output stream buffer.*

• char outBuf [MAX_NETOBJ_SZ]  
  *Buffer to hold converted data before it is written.*

### 8.14.1 Detailed Description

data types

The Dakota::BoStream class is a binary output classes which overloads the \(<<\) operator for all standard data types (int, char, float, etc). The class relies on the built in write methods within the ostream base classes. Dakota::BoStream inherits from the ofstream class. The motivation to develop this class was to replace the Rogue wave class which Dakota historically used for binary I/O. If available, the class utilize rpc/xdr to construct machine independent binary files. These Dakota restart files can be moved between hosts.

### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 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.
8.14 BoStream Class Reference

8.14.2.2 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

8.14.2.3 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.

8.14.3 Member Function Documentation

8.14.3.1 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.

8.14.3.2 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
8.15  COLINApplication Class Reference

Public Member Functions

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

- **~COLINApplication ()**
  
  destructor

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

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

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

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

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

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

Private Member Functions

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

Private Attributes

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

- **ActiveSet activeSet**
  
  copy/conversion of the COLIN request vector

- **bool dakotaModelAsynchFlag**
  
  a flag for asynchronous DAKOTA evaluations
• bool blockingSynch
  
  needed for APPS, to enforce blocking synch despite call of next_eval().

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

• size_t numObjFs
  
  number of objective functions

• size_t numNonlinCons
  
  number of nonlinear constraints

8.15.1 Detailed Description

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

8.15.2 Member Function Documentation

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

launch a function evaluation either synchronously or asynchronously

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

8.15.2.2 void synchronize ()

blocking retrieval of all pending jobs

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

8.15.2.3 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.
8.15.2.4 void map_response (ColinResponse & colin_response, const Response & dakota_response)
[private]

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

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

Wrapper class for optimizers defined using COLIN.

Inheritance diagram for COLINOptimizer:

![Inheritance Diagram]

**Public Member Functions**

- **COLINOptimizer (Model &model)**

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

- **COLINOptimizer (NoDBaseConstructor, Model &model)**
  
  *alternate constructor for Iterator instantiations by name*

- **~COLINOptimizer ()**
  
  *destructor*

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

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

- **template<> bool returns_multiple_points () const**
  
  *return is false. Override to return true if appropriate.*

- **template<> bool returns_multiple_points () const**
  
  *return is false. Override to return true if appropriate.*

- **template<> void set_method_parameters ()**

- **template<> void set_method_parameters ()**

- **template<> void set_method_parameters ()**
• template<> void set_runtime_parameters ()
• template<> void set_method_parameters ()
• template<> void set_method_parameters ()
• template<> void set_method_parameters ()
• template<> void get_final_points ()
• template<> void get_final_points ()
• template<> void get_final_points ()

Protected Member Functions

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

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

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

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

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

• virtual void set_runtime_parameters ()
  
  not available until run time

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

• void resize_final_points (size_t newsize)
  
  resize bestVariablesArray

Protected Attributes

• OptimizerT * optimizer
  
  Pointer to COLIN base optimizer object.

• COLINApplication * application
  
  Pointer to the COLINApplication object.

• colin::OptProblem< ColinPoint > problem
  
  the COLIN problem object
8.16 COLINOptimizer Class Template Reference

- `utilib::RNG * rng`
  
  RNG ptr.

- `bool blockingSynch`
  nonblocking

- `Real solverStartTime`
  Start time for keeping track of time for solver to run.

- `Real solverTime`
  Time taken by solver to run.

8.16.1 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_target` and `max_cpu_time` are mapped into COLIN’s `max_iters`, `max_neval`, `ftol`, `accuracy`, and `max_time` data attributes. An output setting of `verbose` is passed to COLIN’s `set_output()` function and a setting of `debug` activates output of method initialization and sets the COLIN `debug` attribute to 10000. Refer to [Hart, W.E., 2006] for additional information on COLIN objects and controls.

8.16.2 Member Function Documentation

8.16.2.1 `void find_optimum () [inline, virtual]`

Performs the iterations to determine the optimal solution.

find_optimum redefines the Optimizer virtual function to perform the optimization using COLIN. It first sets up the problem data, then executes minimize() on the COLIN optimizer, and finally catalogues the results.

Get the best points from the solver

Implements Optimizer.

8.16.2.2 `void set_standard_method_parameters () [inline, protected]`

sets the standard method parameters shared by all methods
set_standard_method_parameters propagates standard DAKOTA user input to the optimizer.

8.16.2.3  void set_method_parameters ()  [inline]
specialization of set_method_parameters() for DIRECT

8.16.2.4  void set_method_parameters ()  [inline]
specialization of set_method_parameters() for Cobyla

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

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

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

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

8.16.2.9  void set_method_parameters ()  [inline]
specialization of set_method_parameters() for EAminlp

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

- COLINOptimizer.H
8.17 ColinPoint Class Reference

Public Attributes

- std::vector<double> rvec
  
  continuous parameter values

- std::vector<int> ivec
  
  discrete integer parameter values

8.17.1 Detailed Description

A class containing a vector of doubles and integers.
The documentation for this class was generated from the following file:

- COLINAApplication.H
8.18 CollaborativeHybridStrategy Class Reference

optimization and nonlinear least squares methods.

Inheritance diagram for CollaborativeHybridStrategy:

```
Strategy
   |
   v
HybridStrategy
   |
   v
CollaborativeHybridStrategy
```

Public Member Functions

- **CollaborativeHybridStrategy** (ProblemDescDB &problem_db)  
  constructor

- **~CollaborativeHybridStrategy** ()  
  destructor

Protected Member Functions

- **void run_strategy** ()  
  Performs the collaborative hybrid minimization strategy.

- **const Variables & variables_results** () const  
  return the final solution from the collaborative minimization (variables)

- **const Response & response_results** () const  
  return the final solution from the collaborative minimization (response)

Private Attributes

- **String hybridCollabType**  
  abo or hops

- **Variables bestVariables**  
  best variables found in minimization

- **Response bestResponse**  
  best response found in minimization
8.18 CollaborativeHybridStrategy Class Reference

8.18.1 Detailed Description

optimization and nonlinear least squares methods.

This strategy has two approaches to hybrid minimization: (1) agent-based using the ABO framework; (2) nonagent-based using the HOPSPACK framework.

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

- CollaborativeHybridStrategy.H
- CollaborativeHybridStrategy.C
8.19 CommandLineHandler Class Reference

Utility class for managing command line inputs to DAKOTA.

Inheritance diagram for CommandLineHandler::

```
CommandLineHandler
\|-- GetLongOpt
```

Public Member Functions

- **CommandLineHandler ()**
  
  default constructor, requires check_usage() call for parsing

- **CommandLineHandler (int argc, char **argv)**
  
  constructor with parsing

- **~CommandLineHandler ()**
  
  destructor

- **void check_usage (int argc, char **argv)**
  
  Prints a descriptive message and exits the program if incorrect.

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

Private Member Functions

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

- **void output_version (std::ostream &s) const**
  
  outputs the DAKOTA version

8.19.1 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
8.20 CommandShell Class Reference

processes with system calls.

Public Member Functions

- **CommandShell ()**
  
  *constructor*

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

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

- **CommandShell & operator<< (CommandShell &(*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*

Public Attributes

- **const char ∗wd**
  
  *To convey working directory when useWorkdir is true:*

Private Attributes

- **String unixCommand**
  
  *insertions and then executed by flush*
8.20 CommandShell Class Reference

- bool asynchFlag
  flags nonblocking operation (background system calls)

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

8.20.1 Detailed Description

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

8.20.2 Member Function Documentation

8.20.2.1 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
8.21 ConcurrentStrategy Class Reference

Strategy for multi-start iteration or pareto set optimization.

Inheritance diagram for ConcurrentStrategy:

```
Strategy

ConcurrentStrategy
```

Public Member Functions

- **ConcurrentStrategy** (ProblemDescDB &problem_db)
  
  constructor

- **~ConcurrentStrategy**()
  
  destructor

Protected Member Functions

- **void run_strategy**()
  
  settings within the iterator or model.

- **void initialize_iterator** (int job_index)
  
  scheduling function (serve_iterators() or static_schedule_iterators())

- **void pack_parameters_buffer** (MPIPackBuffer &send_buffer, int job_index)
  
  pack a send_buffer for assigning an iterator job to a server

- **void unpack_parameters_buffer** (MPIUnpackBuffer &recv_buffer)
  
  unpack a recv_buffer for accepting an iterator job from the scheduler

- **void pack_results_buffer** (MPIPackBuffer &send_buffer, int job_index)
  
  pack a send_buffer for returning iterator results from a server

- **void unpack_results_buffer** (MPIUnpackBuffer &recv_buffer, int job_index)
  
  unpack a recv_buffer for accepting iterator results from a server

- **void update_local_results** (int job_index)
  
  update local prpResults with current iteration results
Private Member Functions

- void initialize_iterator (const RealVector &param_set)
  \nThe function \texttt{initialize_iterator(int)} is used to update \texttt{userDefinedModel} and \texttt{selectedIterator}

- void print_results () const
  \nThe function prints the concurrent iteration results summary (called by \texttt{run_strategy()})

Private Attributes

- Model userDefinedModel
  \nThe model used by the iterator

- Iterator selectedIterator
  \nThe iterator used by the concurrent strategy

- bool multiStartFlag
  \n  Distinguishes multi-start from Pareto-set

- RealVector initialPt
  \n  Point in the Pareto set strategy

- RealVectorArray parameterSets
  \n  Be performed.

8.21.1 Detailed Description

\textbf{Strategy} for multi-start iteration or Pareto set optimization.

This strategy maintains two concurrent iterator capabilities. First, a general capability for running an iterator multiple times from different starting points is provided (often used for multi-start optimization, but not restricted to optimization). Second, a simple capability for mapping the "Pareto frontier" (the set of optimal solutions in multiobjective formulations) is provided. This Pareto set is mapped through running an optimizer multiple times for different sets of multiobjective weightings.

8.21.2 Member Function Documentation

8.21.2.1 void pack_parameters_buffer (MPIPackBuffer & send_buffer, int job_index) [inline, protected, virtual]

Pack a send_buffer for assigning an iterator job to a server.

This virtual function redefinition is executed on the dedicated master processor for self scheduling. It is not used for peer partitions.

Reimplemented from \texttt{Strategy}.
8.21.2.2  void unpack_parameters_buffer (MPIUnpackBuffer & recv_buffer)  [inline, protected, virtual]
unpack a recv_buffer for accepting an iterator job from the scheduler
This virtual function redefinition is executed on an iterator server for dedicated master self scheduling. It is not
used for peer partitions.
Reimplemented from Strategy.

8.21.2.3  void pack_results_buffer (MPIPackBuffer & send_buffer, int job_index)  [inline, protected, virtual]
pack a send_buffer for returning iterator results from a server
This virtual function redefinition is executed either on an iterator server for dedicated master self scheduling or
on peers 2 through n for static scheduling.
Reimplemented from Strategy.

8.21.2.4  void unpack_results_buffer (MPIUnpackBuffer & recv_buffer, int job_index)  [inline, protected, virtual]
unpack a recv_buffer for accepting iterator results from a server
This virtual function redefinition is executed on an strategy master (either the dedicated master processor for self
scheduling or peer 1 for static scheduling).
Reimplemented from Strategy.
The documentation for this class was generated from the following files:

- ConcurrentStrategy.H
- ConcurrentStrategy.C
CONMINOptimizer Class Reference

Wrapper class for the CONMIN optimization library.

Inheritance diagram for CONMINOptimizer::

```
CONMINOptimizer
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Minimizer
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Optimizer
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Iterator

Public Member Functions

- **CONMINOptimizer (Model &model)**
  
  *standard constructor*

- **CONMINOptimizer (NoDBBaseConstructor, Model &model)**

  *alternate constructor*

- **~CONMINOptimizer ()**

  *destructor*

- **void find_optimum ()**

  *Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- **void derived_initialize_run ()**

  *performs run-time set up*

Private Member Functions

- **void initialize ()**

  *Shared constructor code.*

- **void allocate_workspace ()**

  *Allocates workspace for the optimizer.*
• void **deallocate_workspace** ()

  Releases workspace memory.

• void **allocate_constraints** ()

  Allocates constraint mappings.

**Private Attributes**

• int **conminInfo**

  INFO from CONMIN manual.

• int **printControl**

  IPRINT from CONMIN manual (controls output verbosity).

• int **optimizationType**

  MINMAX from DOT manual (minimize or maximize).

• Real **objFnValue**

  value of the objective function passed to CONMIN

• RealVector **constraintValues**

  array of nonlinear constraint values passed to CONMIN

• int **numConminNlnConstr**

  total number of nonlinear constraints seen by CONMIN

• int **numConminLinConstr**

  total number of linear constraints seen by CONMIN

• int **numConminConstr**

  total number of linear and nonlinear constraints seen by CONMIN

• SizetArray **constraintMappingIndices**

  Response constraints used in computing the CONMIN constraints.

• RealArray **constraintMappingMultipliers**

  the CONMIN constraints.

• RealArray **constraintMappingOffsets**

  CONMIN constraints.

• int **N1**

  Size variable for CONMIN arrays. See CONMIN manual.
8.22 CONMINOptimizer Class Reference

- int N2
  Size variable for CONMIN arrays. See CONMIN manual.

- int N3
  Size variable for CONMIN arrays. See CONMIN manual.

- int N4
  Size variable for CONMIN arrays. See CONMIN manual.

- int N5
  Size variable for CONMIN arrays. See CONMIN manual.

- int NFDG
  Finite difference flag.

- int IPRINT
  Flag to control amount of output data.

- int ITMAX
  Flag to specify the maximum number of iterations.

- double FDCH
  Relative finite difference step size.

- double FDCHM
  Absolute finite difference step size.

- double CT
  Constraint thickness parameter.

- double CDMIN
  Minimum absolute value of CT used during optimization.

- double CTL
  Constraint thickness parameter for linear and side constraints.

- double CTLMIN
  Minimum value of CTL used during optimization.

- double DELFUN
  Relative convergence criterion threshold.

- double DABFUN
  Absolute convergence criterion threshold.

- double * conminDesVars
Array of design variables used by CONMIN (length NI = numdv+2).

- double * conminLowerBnds
  Array of lower bounds used by CONMIN (length NI = numdv+2).

- double * conminUpperBnds
  Array of upper bounds used by CONMIN (length NI = numdv+2).

- double * S
  Internal CONMIN array.

- double * G1
  Internal CONMIN array.

- double * G2
  Internal CONMIN array.

- double * B
  Internal CONMIN array.

- double * C
  Internal CONMIN array.

- int * MS1
  Internal CONMIN array.

- double * SCAL
  Internal CONMIN array.

- double * DF
  Internal CONMIN array.

- double * A
  Internal CONMIN array.

- int * ISC
  Internal CONMIN array.

- int * IC
  Internal CONMIN array.
8.22 CONMINOptimizer Class Reference

8.22.1 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.

8.22.2 Member Data Documentation

8.22.2.1 int conminInfo [private]

INFO from CONMIN manual.

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

8.22.2.2 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.

8.22.2.3 int optimizationType [private]

MINMAX from DOT manual (minimize or maximize).

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

8.22.2.4 RealVector constraintValues [private]

array of nonlinear constraint values passed to CONMIN.

This array must be of nonzero length and must contain only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities and equalities).
8.22.2.5 **SizetArray constraintMappingIndices** [private]

Response constraints used in computing the CONMIN constraints.

The length of the container corresponds to the number of CONMIN constraints, and each entry in the container points to the corresponding DAKOTA constraint.

8.22.2.6 **RealArray constraintMappingMultipliers** [private]

The length of the container corresponds to the number of CONMIN constraints, and each entry in the container stores a multiplier for the DAKOTA constraint identified with constraintMappingIndices. These multipliers are currently +1 or -1.

8.22.2.7 **RealArray constraintMappingOffsets** [private]

The length of the container corresponds to the number of CONMIN constraints, and each entry in the container stores an offset for the DAKOTA constraint identified with constraintMappingIndices. These offsets involve inequality bounds or equality targets, since CONMIN assumes constraint allowables = 0.

8.22.2.8 **int N1** [private]

Size variable for CONMIN arrays. See CONMIN manual.

N1 = number of variables + 2

8.22.2.9 **int N2** [private]

Size variable for CONMIN arrays. See CONMIN manual.

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

8.22.2.10 **int N3** [private]

Size variable for CONMIN arrays. See CONMIN manual.

N3 = Maximum possible number of active constraints.

8.22.2.11 **int N4** [private]

Size variable for CONMIN arrays. See CONMIN manual.

N4 = Maximum(N3,number of variables)
8.22 CONMINOptimizer Class Reference

8.22.2.12 int $N5$ [private]

Size variable for CONMIN arrays. See CONMIN manual. 
$N5 = 2 \times (N4)$

8.22.2.13 double $CT$ [private]

Constraint thickness parameter. 
The value of CT decreases in magnitude during optimization.

8.22.2.14 double* $S$ [private]

Internal CONMIN array. 
Move direction in N-dimensional space.

8.22.2.15 double* $G1$ [private]

Internal CONMIN array. 
Temporary storage of constraint values.

8.22.2.16 double* $G2$ [private]

Internal CONMIN array. 
Temporary storage of constraint values.

8.22.2.17 double* $B$ [private]

Internal CONMIN array. 
Temporary storage for computations involving array $S$.

8.22.2.18 double* $C$ [private]

Internal CONMIN array. 
Temporary storage for use with arrays $B$ and $S$.

8.22.2.19 int* $MS1$ [private]

Internal CONMIN array. 
Temporary storage for use with arrays $B$ and $S$. 
8.22.2.20 **double** `SCAL` [private]

Internal CONMIN array.
Vector of scaling parameters for design parameter values.

8.22.2.21 **double** `DF` [private]

Internal CONMIN array.
Temporary storage for analytic gradient data.

8.22.2.22 **double** `A` [private]

Internal CONMIN array.
Temporary 2-D array for storage of constraint gradients.

8.22.2.23 **int** `ISC` [private]

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

8.22.2.24 **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`
8.23 Constraints Class Reference

Base class for the variable constraints class hierarchy.

Inheritance diagram for Constraints::

```
+-------------------+       +-------------------+
| Constraints       |       | MixedConstraints  |
|                   +-------+-------------------+
| MergedConstraints |
```

Public Member Functions

- **Constraints ()**
  - default constructor

- **Constraints (const ProblemDescDB &prob_db, const std::pair< short, short >&view, const Sizet2DArray &vars_comps)**
  - standard constructor

- **Constraints (const std::pair< short, short >&view, const Sizet2DArray &vars_comps)**
  - alternate constructor for instantiations on the fly

- **Constraints (const Constraints &con)**
  - copy constructor

- virtual ~Constraints ()
  - destructor

- **Constraints operator= (const Constraints &con)**
  - assignment operator

- virtual void write (std::ostream &s) const
  - write a variable constraints object to an std::ostream

- virtual void read (std::istream &s)
  - read a variable constraints object from an std::istream

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

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

- void continuous_lower_bound (const Real &cl_bnd, const size_t &i)
set an active continuous variable lower bound

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

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

- void continuous_upper_bound (const Real &cu_bnd, const size_t &i)
  set an active continuous variable upper bound

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

- void discrete_int_lower_bounds (const IntVector &dil_bnds)
  set the active discrete variable lower bounds

- void discrete_int_lower_bound (const int &dil_bnd, const size_t &i)
  set an active discrete variable lower bound

- const IntVector & discrete_int_upper_bounds () const
  return the active discrete variable upper bounds

- void discrete_int_upper_bounds (const IntVector &diu_bnds)
  set the active discrete variable upper bounds

- void discrete_int_upper_bound (const int &diu_bnd, const size_t &i)
  set an active discrete variable upper bound

- const RealVector & discrete_real_lower_bounds () const
  return the active discrete variable lower bounds

- void discrete_real_lower_bounds (const RealVector &drl_bnds)
  set the active discrete variable lower bounds

- void discrete_real_lower_bound (const Real &drl_bnd, const size_t &i)
  set an active discrete variable lower bound

- const RealVector & discrete_real_upper_bounds () const
  return the active discrete variable upper bounds

- void discrete_real_upper_bounds (const RealVector &dru_bnds)
  set the active discrete variable upper bounds

- void discrete_real_upper_bound (const Real &dru_bnd, const size_t &i)
  set an active discrete variable upper bound
• const RealVector & inactive_continuous_lower_bounds () const
  return the inactive continuous lower bounds

• void inactive_continuous_lower_bounds (const RealVector &icl_bnds)
  set the inactive continuous lower bounds

• const RealVector & inactive_continuous_upper_bounds () const
  return the inactive continuous upper bounds

• void inactive_continuous_upper_bounds (const RealVector &icu_bnds)
  set the inactive continuous upper bounds

• const IntVector & inactive_discrete_int_lower_bounds () const
  return the inactive discrete lower bounds

• void inactive_discrete_int_lower_bounds (const IntVector &idil_bnds)
  set the inactive discrete lower bounds

• const IntVector & inactive_discrete_int_upper_bounds () const
  return the inactive discrete upper bounds

• void inactive_discrete_int_upper_bounds (const IntVector &idiu_bnds)
  set the inactive discrete upper bounds

• const RealVector & inactive_discrete_real_lower_bounds () const
  return the inactive discrete lower bounds

• void inactive_discrete_real_lower_bounds (const RealVector &idrl_bnds)
  set the inactive discrete lower bounds

• const RealVector & inactive_discrete_real_upper_bounds () const
  return the inactive discrete upper bounds

• void inactive_discrete_real_upper_bounds (const RealVector &idru_bnds)
  set the inactive discrete upper bounds

• const RealVector & all_continuous_lower_bounds () const
  returns a single array with all continuous lower bounds

• void all_continuous_lower_bounds (const RealVector &acl_bnds)
  sets all continuous lower bounds using a single array

• void all_continuous_lower_bound (const Real &acl_bnd, const size_t &i)
  set a lower bound within the all continuous lower bounds array
• const RealVector & all_continuous_upper_bounds () const
  returns a single array with all continuous upper bounds

• void all_continuous_upper_bounds (const RealVector &acu_bnds)
  sets all continuous upper bounds using a single array

• void all_continuous_upper_bound (const Real &acu_bnd, const size_t &i)
  sets an upper bound within the all continuous upper bounds array

• const IntVector & all_discrete_int_lower_bounds () const
  returns a single array with all discrete lower bounds

• void all_discrete_int_lower_bounds (const IntVector &adil_bnds)
  sets all discrete lower bounds using a single array

• void all_discrete_int_lower_bound (const int &adil_bnd, const size_t &i)
  sets a lower bound within the all discrete lower bounds array

• const IntVector & all_discrete_int_upper_bounds () const
  returns a single array with all discrete upper bounds

• void all_discrete_int_upper_bounds (const IntVector &adiu_bnds)
  sets all discrete upper bounds using a single array

• void all_discrete_int_upper_bound (const int &adiu_bnd, const size_t &i)
  sets an upper bound within the all discrete upper bounds array

• const RealVector & all_discrete_real_lower_bounds () const
  returns a single array with all discrete lower bounds

• void all_discrete_real_lower_bounds (const RealVector &adrl_bnds)
  sets all discrete lower bounds using a single array

• void all_discrete_real_lower_bound (const Real &adrl_bnd, const size_t &i)
  sets a lower bound within the all discrete lower bounds array

• const RealVector & all_discrete_real_upper_bounds () const
  returns a single array with all discrete upper bounds

• void all_discrete_real_upper_bounds (const RealVector &adru_bnds)
  sets all discrete upper bounds using a single array

• void all_discrete_real_upper_bound (const Real &adru_bnd, const size_t &i)
  sets an upper bound within the all discrete upper bounds array

• size_t num_linear_ineq_constraints () const
return the number of linear inequality constraints

- size_t num_linear_eq_constraints() const
  
  return the number of linear equality constraints

- const RealMatrix & linear_ineq_constraint_coeffs() const
  
  return the linear inequality constraint coefficients

- void linear_ineq_constraint_coeffs(const RealMatrix &lin_ineq_coeffs)
  
  set the linear inequality constraint coefficients

- const RealVector & linear_ineq_constraint_lower_bounds() const
  
  return the linear inequality constraint lower bounds

- void linear_ineq_constraint_lower_bounds(const RealVector &lin_ineq_l_bnds)
  
  set the linear inequality constraint lower bounds

- const RealVector & linear_ineq_constraint_upper_bounds() const
  
  return the linear inequality constraint upper bounds

- void linear_ineq_constraint_upper_bounds(const RealVector &lin_ineq_u_bnds)
  
  set the linear inequality constraint upper bounds

- const RealMatrix & linear_eq_constraint_coeffs() const
  
  return the linear equality constraint coefficients

- void linear_eq_constraint_coeffs(const RealMatrix &lin_eq_coeffs)
  
  set the linear equality constraint coefficients

- const RealVector & linear_eq_constraint_targets() const
  
  return the linear equality constraint targets

- void linear_eq_constraint_targets(const RealVector &lin_eq_targets)
  
  set the linear equality constraint targets

- size_t num_nonlinear_ineq_constraints() const
  
  return the number of nonlinear inequality constraints

- size_t num_nonlinear_eq_constraints() const
  
  return the number of nonlinear equality constraints

- const RealVector & nonlinear_ineq_constraint_lower_bounds() const
  
  return the nonlinear inequality constraint lower bounds

- void nonlinear_ineq_constraint_lower_bounds(const RealVector &lnl_ineq_l_bnds)
  
  set the nonlinear inequality constraint lower bounds
• const RealVector & nonlinear_ineq_constraint_upper_bounds () const
  return the nonlinear inequality constraint upper bounds

• void nonlinear_ineq_constraint_upper_bounds (const RealVector &nln_ineq_u_bnds)
  set the nonlinear inequality constraint upper bounds

• const RealVector & nonlinear_eq_constraint_targets () const
  return the nonlinear equality constraint targets

• void nonlinear_eq_constraint_targets (const RealVector &nln_eq_targets)
  set the nonlinear equality constraint targets

• Constraints copy () const
  for use when a deep copy is needed (the representation is _not_ shared)

• void reshape (const size_t &num_nln_ineq_cons, const size_t &num_nln_eq_cons, const size_t &num_-lin_ineq_cons, const size_t &num_lin_eq_cons, const Sizet2DArray &vars_comps)
  Constraints hierarchy.

• void reshape (const size_t &num_nln_ineq_cons, const size_t &num_nln_eq_cons, const size_t &num_-lin_ineq_cons, const size_t &num_lin_eq_cons)
  Constraints hierarchy.

• void reshape (const Sizet2DArray &vars_comps)
  reshape the lower/upper bound arrays within the Constraints hierarchy

• void inactive_view (short view2)
  sets the inactive view based on higher level (nested) context

• bool is_null () const
  function to check constraintsRep (does this envelope contain a letter)

**Protected Member Functions**

• Constraints (BaseConstructor, const ProblemDescDB &problem_db, const std::pair< short, short > &view, const Sizet2DArray &vars_comps)
  derived class constructors - Coplien, p. 139)

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

• virtual void reshape_rep ()
  Used by reshape(Sizet2DArray&) to rehape the contents of a letter class.
• virtual void build_active_views ()
  construct active views of all variables bounds arrays

• virtual void build_inactive_views ()
  construct inactive views of all variables bounds arrays

• void build_views ()
  construct active/inactive views of all variables arrays

• void manage_linear_constraints (const ProblemDescDB &problem_db)
  coefficient input to matrices, and assign defaults

Protected Attributes

• std::pair< short, short > variablesView
  from the corresponding Variables object.

• const Sizet2DArray * variablesComponents
  is a pointer to the instance from the corresponding Variables object.

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

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

• IntVector allDiscreteIntLowerBnds
  discrete state variable types (all view).

• IntVector allDiscreteIntUpperBnds
  discrete state variable types (all view).

• RealVector allDiscreteRealLowerBnds
  discrete state variable types (all view).

• RealVector allDiscreteRealUpperBnds
  discrete state variable types (all view).

• size_t numNonlinearIneqCons
  number of nonlinear inequality constraints

• size_t numNonlinearEqCons
  number of nonlinear equality constraints

• RealVector nonlinearIneqConLowerBnds
nonlinear inequality constraint lower bounds

- RealVector nonlinearIneqConUpperBnds
  nonlinear inequality constraint upper bounds

- RealVector nonlinearEqConTargets
  nonlinear equality constraint targets

- size_t numLinearIneqCons
  number of linear inequality constraints

- size_t numLinearEqCons
  number of linear equality constraints

- RealMatrix linearIneqConCoeffs
  linear inequality constraint coefficients

- RealMatrix linearEqConCoeffs
  linear equality constraint coefficients

- RealVector linearIneqConLowerBnds
  linear inequality constraint lower bounds

- RealVector linearIneqConUpperBnds
  linear inequality constraint upper bounds

- RealVector linearEqConTargets
  linear equality constraint targets

- RealVector continuousLowerBnds
  the active continuous lower bounds array view

- RealVector continuousUpperBnds
  the active continuous upper bounds array view

- IntVector discreteIntLowerBnds
  the active discrete lower bounds array view

- IntVector discreteIntUpperBnds
  the active discrete upper bounds array view

- RealVector discreteRealLowerBnds
  the active discrete lower bounds array view

- RealVector discreteRealUpperBnds
  the active discrete upper bounds array view
- **RealVector** `inactiveContinuousLowerBnds`
  *the inactive continuous lower bounds array view*

- **RealVector** `inactiveContinuousUpperBnds`
  *the inactive continuous upper bounds array view*

- **IntVector** `inactiveDiscreteIntLowerBnds`
  *the inactive discrete lower bounds array view*

- **IntVector** `inactiveDiscreteIntUpperBnds`
  *the inactive discrete upper bounds array view*

- **RealVector** `inactiveDiscreteRealLowerBnds`
  *the inactive discrete lower bounds array view*

- **RealVector** `inactiveDiscreteRealUpperBnds`
  *the inactive discrete upper bounds array view*

### Private Member Functions

- **Constraints** ∗ `get_constraints`
  (const `ProblemDescDB` &`problem_db`, const std::pair<short, short> &`view`,
   const `Sizet2DArray` &`vars_comps`)
  *appropriate derived type.*

- **Constraints** ∗ `get_constraints`
  (const std::pair<short, short> &`view`) const
  *derived type.*

### Private Attributes

- **Constraints** ∗ `constraintsRep`
  *pointer to the letter (initialized only for the envelope)*

- int `referenceCount`
  *number of objects sharing constraintsRep*

### 8.23.1 Detailed Description

Base class for the variable constraints class hierarchy.

The Constraints class is the base class for the class hierarchy managing bound, linear, and nonlinear constraints. Using the variable lower and upper bounds arrays from the input specification, different derived classes define
different views of this data. The linear and nonlinear constraint data is consistent in all views and is managed at the base class level. For memory efficiency and enhanced polymorphism, the variable constraints hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Constraints) serves as the envelope and one of the derived classes (selected in Constraints::get_constraints()) serves as the letter.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 Constraints ()

default constructor

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

8.23.2.2 Constraints (const ProblemDescDB & problem_db, const std::pair<short, short> & view, const Sizet2DArray & vars_comps)

standard constructor

The envelope constructor only needs to extract enough data to properly execute get_constraints, since the constructor overloaded with BaseConstructor builds the actual base class data inherited by the derived classes.

8.23.2.3 Constraints (const std::pair<short, short> & view, const Sizet2DArray & vars_comps)

alternate constructor for instantiations on the fly

Envelope constructor for instantiations on the fly. This constructor executes get_constraints(view), which invokes the default derived/base constructors, followed by a reshape() based on vars_comps.

8.23.2.4 Constraints (const Constraints & con)

copy constructor

Copy constructor manages sharing of constraintsRep and incrementing of referenceCount.

8.23.2.5 ~Constraints () [virtual]

destructor

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

8.23.2.6 Constraints (BaseConstructor, const ProblemDescDB & problem_db, const std::pair<short, short> & view, const Sizet2DArray & vars_comps) [protected]

derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all derived classes. `get_constraints()` instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling `get_constraints()` again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Constraints).

### 8.23.3 Member Function Documentation

#### 8.23.3.1 `Constraints operator= (const Constraints & con)`

Assignment operator


#### 8.23.3.2 `Constraints copy () const`

for use when a deep copy is needed (the representation is _not_ shared)

Deep copies are used for history mechanisms such as bestVariables and data_pairs since these must catalogue copies (and should not change as the representation within currentVariables changes).

#### 8.23.3.3 `void reshape (const size_t & num_nln_ineq_cons, const size_t & num_nln_eq_cons, const size_t & num_lin_ineq_cons, const size_t & num_lin_eq_cons)`

Resizes the linear and nonlinear constraint arrays at the base class. Does NOT currently resize the derived bounds arrays.

#### 8.23.3.4 `void reshape (const Sizet2DArray & vars_comps)`

reshape the lower/upper bound arrays within the Constraints hierarchy

Resizes the derived bounds arrays.

#### 8.23.3.5 `void build_views () [inline, protected]`

construct active/inactive views of all variables arrays

= EMPTY)

= EMPTY)

#### 8.23.3.6 `void manage_linear_constraints (const ProblemDescDB & problem_db) [protected]`

coefficient input to matrices, and assign defaults

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

8.23.3.7 Constraints * get_constraints (const ProblemDescDB & problem_db, const std::pair< short, short > & view, const Sizet2DArray & vars_comps) [private]

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

8.23.3.8 Constraints * get_constraints (const std::pair< short, short > & view) const [private]

derived type.
Initializes constraintsRep to the appropriate derived type, as given by the variables view. The default derived class constructors are invoked.

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

- DakotaConstraints.H
- DakotaConstraints.C
8.24  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
}

  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)
  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 matching

- RStatus getStatus ()
  Get status.

- const std::string & getStatusMsg ()
  Get status message.

- void clearErrors ()
  Clear all errors.

- const std::string & getRe ()
  Return regular expression pattern.

- bool split (const std::string &str, std::vector<std::string> &all_matches)
  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.

8.24.2 Member Enumeration Documentation

8.24.2.1 enum RStatus

occurs with this implementation.

Enumerator:

- **GOOD** Success - no errors.
- **EXP_TOO_BIG** Regular expression is too big to be compiled.
- **OUT_OF_MEM** out of space (memory)
- **TOO_MANY_PAR** too many () parenteses
- **UNMATCH_PAR** unmatched () parenteses
- **STARPLUS_EMPTY** ++ operand could be empty
- **STARPLUS_NESTED** nested *?
**INDEX_RANGE**  invalid [] range

**INDEX_MATCH**  unmatched []

**STARPLUS NOTHING**  ?++ follows nothing

**TRAILING**  trailing \

**INT_ERROR**  junk on end, "internal urp", "internal disaster"

**BAD_PARAM**  NULL parameter.

**BAD_OPCODE**  corrupted opcode

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

- CtelRegExp.H
- CtelRegExp.C
8.25 DataFitSurrModel Class Reference

data fit surrogates (global and local)

Inheritance diagram for DataFitSurrModel::

```
Model
    SurrogateModel
    DataFitSurrModel
```

Public Member Functions

- `DataFitSurrModel (ProblemDescDB &problem_db)`
  constructor

- `DataFitSurrModel (Iterator &dace_iterator, Model &actual_model, const std::pair< short, short > &view, const Sizet2DArray &vars_comps, const ActiveSet &set, const String &approx_type, const UShortArray &approx_order, const String &corr_type, short corr_order, const String &sample_reuse)`
  alternate constructor for instantiations on the fly

- `~DataFitSurrModel ()`
  destructor

Protected Member Functions

- `void derived_compute_response (const ActiveSet &set)`
  portion of `compute_response()` specific to `DataFitSurrModel`

- `void derived_asynch_compute_response (const ActiveSet &set)`
  portion of `asynch_compute_response()` specific to `DataFitSurrModel`

- `const IntResponseMap & derived_synchronize ()`
  portion of `synchronize()` specific to `DataFitSurrModel`

- `const IntResponseMap & derived_synchronize_nowait ()`
  portion of `synchronize_nowait()` specific to `DataFitSurrModel`

- `Iterator & subordinate_iterator ()`
  return `daceIterator`
• Model & surrogate_model ()
  return this model instance

• Model & truth_model ()
  return actualModel

• void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  return actualModel (and optionally its sub-models)

• void update_from_subordinate_model (bool recurse_flag=true)
  pass request to actualModel if recursing and then update from it

• Interface & interface ()
  return approxInterface

• void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  squares terms and optionally recurses into actualModel

• void surrogate_bypass (bool bypass_flag)
  any lower-level surrogates.

• void surrogate_function_indices (const IntSet &surr_fn_indices)
  and ApproximationInterface::approxFnIndices

• void build_approximation ()
  daceIterator/actualModel to generate new data points

• bool build_approximation (const Variables &vars, const Response &response)
  augment the vars/response anchor point

• void update_approximation (const Variables &vars, const Response &response, bool rebuild_flag)
  approximation if requested

• void update_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  approximation if requested

• void append_approximation (const Variables &vars, const Response &response, bool rebuild_flag)
  requested (requests forwarded to approxInterface)

• void append_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  rebuilds it if requested (requests forwarded to approxInterface)

• Array< Approximation > & approximations ()
  retrieve the set of Approximations from approxInterface
• const RealVectorArray & approximation_coefficients ()
  (request forwarded to approxInterface)

• void approximation_coefficients (const RealVectorArray &approx_coeffs)
  (request forwarded to approxInterface)

• void print_coefficients (std::ostream &s, size_t index) const
  (request forwarded to approxInterface)

• const RealVector & approximation_variances (const RealVector &c_vars)
  (request forwarded to approxInterface)

• const List< SurrogateDataPoint > & approximation_data (size_t index)
  (request forwarded to approxInterface)

• void component_parallel_mode (short mode)
  update component parallel mode for supporting parallelism in actualModel

• void derived_init_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  set up actualModel for parallel operations

• void derived_init_serial ()
  set up actualModel for serial operations.

• void derived_set_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  set active parallel configuration within actualModel

• void derived_free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  (request forwarded to actualModel)

• void serve ()
  Completes when a termination message is received from stop_servers().

• void stop_servers ()
  when DataFitSurrModel iteration is complete.

• void inactive_view (short view, bool recurse_flag=true)
  context and optionally recurse into actualModel

• const String & interface_id () const
  return the approxInterface identifier

• int evaluation_id () const
  return the current evaluation id for the DataFitSurrModel
• void set_evaluation_reference ()
  (request forwarded to approxInterface and actualModel)

• void fine_grained_evaluation_counters ()
  and actualModel

• void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  (request forwarded to approxInterface and actualModel)

Private Member Functions

• void derived_synchronize_approx (const IntResponseMap &approx_resp_map, IntResponseMap &approx_resp_map_rekey)
  derived_synchronize() and derived_synchronize_nowait()

• void update_global ()
  Updates fit arrays for global approximations.

• void update_local_multipoint ()
  Updates fit arrays for local or multipoint approximations.

• void build_global ()
  Builds a global approximation using daceIterator.

• void build_local_multipoint ()
  Builds a local or multipoint approximation using actualModel.

• void update_actual_model ()
  update actualModel with data from current variables/labels/bounds/targets

• void update_from_actual_model ()
  update current variables/labels/bounds/targets with data from actualModel

• bool inside (const RealVector &c_vars, const IntVector &di_vars, const RealVector &dr_vars)
  [d_l_bnds,d_u_bnds]

Private Attributes

• int surrModelEvals
  derived_async_compute_response()

• String sampleReuse
(default if samples file), or none (default if no samples file)

- **String sampleReuseFile**
  
  file name for samples_file specification

- **VariablesList reuseFileVars**
  
  array of variables sets read from the samples_file

- **ResponseList reuseFileResponses**
  
  array of response sets read from the samples_file

- **Interface approxInterface**
  
  (required for both global and local)

- **Model actualModel**
  
  (optional for global, required for local)

- **Iterator daceIterator**
  
  (optional for global since restart data may also be used)

### 8.25.1 Detailed Description

Data fit surrogates (global and local)

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

### 8.25.2 Member Function Documentation

#### 8.25.2.1 void derived_compute_response (const ActiveSet & set)  [protected, virtual]

portion of compute_response() specific to DataFitSurrModel

Compute the response synchronously using actualModel, approxInterface, or both (mixed case). For the approxInterface portion, build the approximation if needed, evaluate the approximate response, and apply correction (if active) to the results.

Reimplemented from Model.

#### 8.25.2.2 void derived_asynch_compute_response (const ActiveSet & set)  [protected, virtual]

portion of asynch_compute_response() specific to DataFitSurrModel
Compute the response asynchronously using actualModel, approxInterface, or both (mixed case). For the approxInterface portion, build the approximation if needed and evaluate the approximate response in a quasi-asynchronous approach (ApproximationInterface::map() performs the map synchronously and bookkeeps the results for return in derived_synchronize() below).

Reimplemented from Model.

8.25.2.3  **const IntResponseMap & derived_synchronize ()**  [protected, virtual]

portion of synchronize() specific to DataFitSurrModel

Blocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case). For the approxInterface portion, apply correction (if active) to each response in the array. derived_synchronize() is designed for the general case where derived_asynch_compute_response() may be inconsistent in its use of actual evaluations, approximate evaluations, or both.

Reimplemented from Model.

8.25.2.4  **const IntResponseMap & derived_synchronize_nowait ()**  [protected, virtual]

portion of synchronize_nowait() specific to DataFitSurrModel

Nonblocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case). For the approxInterface portion, apply correction (if active) to each response in the array. derived_synchronize_nowait() is designed for the general case where derived_asynch_compute_response() may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from Model.

8.25.2.5  **void build_approximation ()**  [protected, virtual]

daceIterator/actualModel to generate new data points

This function constructs a new approximation, discarding any previous data. It constructs any required currentPoints and does not define an anchorPoint.

Reimplemented from Model.

8.25.2.6  **bool build_approximation (const Variables & vars, const Response & response)**  [protected, virtual]

augment the vars/response anchor point

This function constructs a new approximation, discarding any previous data. It uses the passed data to populate the anchorPoint and constructs any required currentPoints.

Reimplemented from Model.
8.25.2.7  void update_approximation (const Variables & vars, const Response & response, bool rebuild_flag) [protected, virtual]

approximation if requested
This function populates/replaces Approximation::anchorPoint and rebuilds the approximation, if requested. It does not clear other data (i.e., Approximation::currentPoints) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.
Reimplemented from Model.

8.25.2.8  void update_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array, bool rebuild_flag) [protected, virtual]

approximation if requested
This function populates/replaces Approximation::currentPoints and rebuilds the approximation, if requested. It does not clear other data (i.e., Approximation::anchorPoint) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.
Reimplemented from Model.

8.25.2.9  void append_approximation (const Variables & vars, const Response & response, bool rebuild_flag) [protected, virtual]

requested (requests forwarded to approxInterface)
This function appends one point to Approximation::currentPoints and rebuilds the approximation, if requested. It does not modify other data (i.e., Approximation::anchorPoint) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.
Reimplemented from Model.

8.25.2.10 void append_approximation (const VariablesArray & vars_array, const ResponseArray & resp_array, bool rebuild_flag) [protected, virtual]

rebuilds it if requested (requests forwarded to approxInterface)
This function appends multiple points to Approximation::currentPoints and rebuilds the approximation, if requested. It does not modify other data (i.e., Approximation::anchorPoint) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.
Reimplemented from Model.
8.25.2.11  void derived_init_communicators (const int & max_iterator_concurrency, bool recurse_flag = true) [inline, protected, virtual]

set up actualModel for parallel operations
asynchronous flags need to be initialized for the sub-models. In addition, max_iterator_concurrency is the outer
level iterator concurrency, not the DACE concurrency that actualModel will see, and recomputing the message_-lengths on the sub-model is probably not a bad idea either. Therefore, recompute everything on actualModel using
init_communicators.
Reimplemented from Model.

8.25.2.12  int evaluation_id () const [inline, protected, virtual]

return the current evaluation id for the DataFitSurrModel
return the DataFitSurrModel evaluation count. Due to possibly intermittent use of surrogate bypass, this is not the
same as either the approxInterface or actualModel model evaluation counts. It also does not distinguish duplicate
evals.
Reimplemented from Model.

8.25.2.13  void build_global () [private]

Builds a global approximation using daceIterator.
Determine sample points to use in building the approximation and then evaluate them on actualModel using
daceIterator. Any changes to the bounds should be performed by setting them at a higher level (e.g., SurrBased-
OptStrategy).

8.25.2.14  void build_local_multipoint () [private]

Builds a local or multipoint approximation using actualModel.
Evaluate the value, gradient, and possibly Hessian needed for a local or multipoint approximation using actual-
Model.

8.25.2.15  void update_actual_model () [private]

update actualModel with data from current variables/labels/bounds/targets
Update variables and constraints data within actualModel using values and labels from currentVariables and
bound/linear/nonlinear constraints from userDefinedConstraints.

8.25.2.16  void update_from_actual_model () [private]

update current variables/labels/bounds/targets with data from actualModel
Update values and labels in currentVariables and bound/linear/nonlinear constraints in userDefinedConstraints
from variables and constraints data within actualModel.
8.25.3 Member Data Documentation

8.25.3.1 Model actualModel [private]

(optional for global, required for local)

actualModel is unrestricted in type; arbitrary nestings are possible.

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

- DataFitSurrModel.H
- DataFitSurrModel.C
Handle class for interface specification data.

Public Member Functions

- **DataInterface ()**
  
  constructor

- **DataInterface (const DataInterface &)**
  
  copy constructor

- **~DataInterface ()**
  
  destructor

- **DataInterface & operator=(const DataInterface &)**
  
  assignment operator

- **void write (std::ostream &s) const**
  
  write a DataInterface object to an std::ostream

- **void read (MPIUnpackBuffer &s)**

  read a DataInterface object from a packed MPI buffer

- **void write (MPIPackBuffer &s) const**

  write a DataInterface object to a packed MPI buffer

Public Attributes

- **DataInterfaceRep * dataIfaceRep**

  pointer to the body (handle-body idiom)

8.26.1 Detailed Description

Handle class for interface specification data.

The DataInterface class is used to provide a memory management handle for the data in DataInterfaceRep. It is populated by IDRProblemDescDB::interface_kwhandler() and is queried by the ProblemDescDB::get_-<datatype>() functions. A list of DataInterface objects is maintained in ProblemDescDB::dataInterfaceList, one for each interface specification in an input file.

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

- DataInterface.H
- DataInterface.C
8.27 DataMethod Class Reference

Handle class for method specification data.

Public Member Functions

- DataMethod()
  
  constructor

- DataMethod (const DataMethod &)
  
  copy constructor

- ~DataMethod()
  
  destructor

- DataMethod & operator= (const DataMethod &)
  
  assignment operator

- void write (std::ostream &s) const
  
  write a DataMethod object to an std::ostream

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

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

Public Attributes

- DataMethodRep * dataMethodRep
  
  pointer to the body (handle-body idiom)

8.27.1 Detailed Description

Handle class for method specification data.

The DataMethod class is used to provide a memory management handle for the data in DataMethodRep. It is populated by IDRProblemDescDB::method_khandler() and is queried by the ProblemDescDB::get_<datatype> () functions. A list of DataMethod objects is maintained in ProblemDescDB::dataMethodList, one for each method specification in an input file.

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

- DataMethod.H
- DataMethod.C
8.28 DataMethodRep Class Reference

Body class for method specification data.

Public Attributes

- **String idMethod**
  
  *the id_method specification in MethodIndControl*

- **String modelPointer**
  
  *(from the model_pointer specification in MethodIndControl)*

- **short methodOutput**
  
  *(from the output specification in MethodIndControl)*

- **int maxIterations**
  
  *max_iterations specification in MethodIndControl*

- **int maxFunctionEvaluations**
  
  *the max_function_evaluations specification in MethodIndControl*

- **bool speculativeFlag**
  
  *(from the speculative specification in MethodIndControl)*

- **Real convergenceTolerance**
  
  *convergence_tolerance specification in MethodIndControl*

- **Real constraintTolerance**
  
  *constraint_tolerance specification in MethodIndControl*

- **bool methodScaling**
  
  *MethodIndControl*

- **RealVector linearIneqConstraintCoeffs**
  
  *MethodIndControl*.  

- **RealVector linearIneqLowerBnds**
  
  *linear_inequality_lower_bounds specification in MethodIndControl*

- **RealVector linearIneqUpperBnds**
  
  *linear_inequality_upper_bounds specification in MethodIndControl*

- **StringArray linearIneqScaleTypes**
  
  *linear_inequality_scale_types specification in MethodIndControl*
- RealVector `linearIneqScales`
  
  `linear_inequality_scales` specification in `MethodIndControl`

- RealVector `linearEqConstraintCoeffs`

  `MethodIndControl`.

- RealVector `linearEqTargets`

  `linear_equality_targets` specification in `MethodIndControl`

- StringArray `linearEqScaleTypes`

  `linear_equality_scale_types` specification in `MethodIndControl`

- RealVector `linearEqScales`

  `linear_equality_scales` specification in `MethodIndControl`

- String `methodName` or parameter study methods

- String `subMethodName`

  (from the `sub_method_name` specification in SBL/SGB)

- String `subMethodPointer`

  method (from the `sub_method_pointer` specification in SBL/SGB)

- int `surrBasedLocalSoftConvLimit`

  `soft_convergence_limit` specification in `MethodSBL`

- bool `surrBasedLocalLayerBypass`

  layerings in evaluating truth response values in SBL

- Real `surrBasedLocalTRInitSize`

  distance (upper bound - lower bound) for each variable

- Real `surrBasedLocalTRMinSize`

  regions

- Real `surrBasedLocalTRContractTrigger`

  this value ("eta_1" in the Conn-Gould-Toint trust region book)

- Real `surrBasedLocalTRExpandTrigger`

  value ("eta_2" in the Conn-Gould-Toint trust region book)

- Real `surrBasedLocalTRContract`

  (from the `contraction_factor` specification in `MethodSBL`)

- Real `surrBasedLocalTRExpand`
• short **surrBasedLocalSubProbObj**
  
  LAGRANGIAN\_OBJECTIVE, or AUGMENTED\_LAGRANGIAN\_OBJECTIVE.

• short **surrBasedLocalSubProbCon**

  LINEARIZED\_CONSTRAINTS, or ORIGINAL\_CONSTRAINTS.

• short **surrBasedLocalMeritFn**

  BASIC\_LAGRANGIAN, or AUGMENTED\_LAGRANGIAN.

• short **surrBasedLocalAcceptLogic**

  SBL iterate acceptance logic: TR\_RATIO or FILTER.

• short **surrBasedLocalConstrRelax**

  SBL constraint relaxation method: NO\_RELAX or HOMOTOPY.

• bool **surrBasedGlobalReplacePts**

  next surrogate is based in the surrogate\_based\_global strategy.

• **String minMaxType**

  the optimization\_type specification in MethodDOTDC

• **String dlDetails**

  string of options for a dynamically linked solver

• int **verifyLevel**

  the verify\_level specification in MethodNPSOLDC

• **Real functionPrecision**

  the function\_precision specification in MethodNPSOLDC

• **Real lineSearchTolerance**

  the linesearch\_tolerance specification in MethodNPSOLDC

• **Real absConvTol**

  absolute function convergence tolerance

• **Real xConvTol**

  x-convergence tolerance

• **Real singConvTol**

  singular convergence tolerance

• **Real singRadius**

  radius for singular convergence test
- Real `falseConvTol`
  *false-convergence tolerance*

- Real `initTRRadius`
  *initial trust radius*

- int `covarianceType`
  *kind of covariance required*

- bool `regressDiag`
  *whether to print the regression diagnostic vector*

- String `searchMethod`
  *interior-point methods in MethodOPTPPDC*

- Real `gradientTolerance`
  *the gradient_tolerance specification in MethodOPTPPDC*

- Real `maxStep`
  *the max_step specification in MethodOPTPPDC*

- String `meritFn`
  *interior-point methods in MethodOPTPPDC*

- String `centralPath`
  *methods in MethodOPTPPDC*

- Real `stepLenToBoundary`
  *interior-point methods in MethodOPTPPDC*

- Real `centeringParam`
  *interior-point methods in MethodOPTPPDC*

- int `searchSchemeSize`
  *MethodOPTPPDC*

- Real `initStepLength`
  *MethodAPPSDC*

- Real `contractStepLength`
  *MethodAPPSDC*

- Real `threshStepLength`
  *MethodAPPSDC*
8.28 DataMethodRep Class Reference

- String evalSynchronize
  MethodAPPSDC

- String meritFunction
  MethodAPPSDC

- Real constrPenalty
  MethodAPPSDC

- Real smoothFactor
  MethodAPPSDC

- String evalSynchronization
  methods in MethodCOLINYP and MethodAPPS

- Real constraintPenalty
  MethodCOLINYSW and MethodCOLINYEA

- bool constantPenalty
  MethodCOLINYP and MethodCOLINYSW

- Real globalBalanceParam
  MethodCOLINYDIR

- Real localBalanceParam
  MethodCOLINYDIR

- Real maxBoxSize
  the max_boxsize_limit for the DIRECT method in MethodCOLINYDIR

- Real minBoxSize
  and MethodNCSUDC

- String boxDivision
  the DIRECT method in MethodCOLINYDIR

- bool mutationAdaptive
  MethodCOLINYEA

- bool showMiscOptions
  the show_misc_options specification in MethodCOLINYDC

- StringArray miscOptions
  the misc_options specification in MethodCOLINYDC

- Real solnTarget
the solution_target specification in MethodCOLINYDC

- Real crossoverRate
  the crossover_rate specification for EA methods in MethodCOLINYEA

- Real mutationRate
  the mutation_rate specification for EA methods in MethodCOLINYEA

- Real mutationScale
  the mutation_scale specification for EA methods in MethodCOLINYEA

- Real mutationMinScale
  MethodCOLINYEA

- Real initDelta
  MethodCOLINYSW

- Real threshDelta
  MethodCOLINYSW

- Real contractFactor
  MethodAPPS, MethodCOLINYS, and MethodCOLINYSW

- int newSolnsGenerated
  in MethodCOLINYEA

- int numberRetained
  MethodCOLINYEA.

- bool expansionFlag
  MethodAPPS, MethodCOLINYS, and MethodCOLINYSW

- int expandAfterSuccess
  MethodCOLINYS and MethodCOLINYSW

- int contractAfterFail
  MethodCOLINYSW

- int mutationRange
  MethodCOLINYEA

- int totalPatternSize
  MethodCOLINYS

- bool randomizeOrderFlag
  the stochastic specification for the PS method in MethodCOLINYS


- **String** selectionPressure
  
  *the fitness_type specification for EA methods in MethodCOLINYEA*

- **String** replacementType
  
  MethodCOLINYEA

- **String** crossoverType
  
  *the crossover_type specification for EA methods in MethodCOLINYEA*

- **String** mutationType
  
  *the mutation_type specification for EA methods in MethodCOLINYEA*

- **String** exploratoryMoves
  
  MethodCOLINYPSS

- **String** patternBasis
  
  MethodAPPS and MethodCOLINYPSS

- **size_t** numCrossPoints
  
  *The number of crossover points or multi-point schemes.*

- **size_t** numParents
  
  *The number of parents to use in a crossover operation.*

- **size_t** numOffspring
  
  *The number of children to produce in a crossover operation.*

- **String** fitnessType
  
  *the fitness assessment operator to use.*

- **String** convergenceType
  
  *The means by which this JEGA should converge.*

- **Real** percentChange
  
  *for a fitness tracker converger.*

- **size_t** numGenerations
  
  *tracker converger should track.*

- **Real** fitnessLimit
  
  *below_limit selector).*

- **Real** shrinkagePercent
  
  *must take place on each call to the selector (0, 1).*
• **String nichingType**  
  *The niching type.*

• **RealVector nicheVector**  
  *The discretization percentage along each objective.*

• **String postProcessorType**  
  *The post processor type.*

• **RealVector distanceVector**  
  *The discretization percentage along each objective.*

• **String initializationType**  
  *The means by which the JEGA should initialize the population.*

• **String flatFile**  
  *The filename to use for initialization.*

• **String logFile**  
  *The filename to use for logging.*

• **int populationSize**  
  *MethodCOLINYEA.*

• **bool printPopFlag**  
  *at each generation*

• **Real volBoxSize**  
  *the volume_boxsize_limit for the DIRECT method in MethodNCSUDC*

• **String daceMethod**  
  *dace specification in MethodDDACE)*

• **int numSymbols**  
  *the symbols specification for DACE methods*

• **bool mainEffectsFlag**  
  *in MethodDDACE)*

• **bool latinizeFlag**  
  *MethodFSUDACE*

• **bool volQualityFlag**  
  *and CVT methods in MethodFSUDACE)*

• **bool varBasedDecompFlag**
and CVT methods in MethodFSUDACE)

- **IntVector sequenceStart**
  the sequenceStart specification in MethodFSUDACE

- **IntVector sequenceLeap**
  the sequenceLeap specification in MethodFSUDACE

- **IntVector primeBase**
  the primeBase specification in MethodFSUDACE

- **int numTrials**
  the numTrials specification in MethodFSUDACE

- **String trialType**
  the trial_type specification in MethodFSUDACE

- **int randomSeed**
  the seed specification for COLINY, NonD, & DACE methods

- **int numSamples**
  the samples specification for NonD & DACE methods

- **bool fixedSeedFlag**
  stencil/pattern throughout a strategy with repeated sampling.

- **bool fixedSequenceFlag**
  stencil/pattern throughout a strategy with repeated sampling.

- **int previousSamples**
  the number of previous samples when augmenting a LHS sample

- **String rngName**
  the basic random-number generator for NonD

- **short expansionType**
  ASKEY_U or STD_NORMAL_U based on input keywords askey or wiener.

- **int expansionTerms**
  the expansion_terms specification in MethodNonDPCE

- **UShortArray expansionOrder**
  the expansion_order specification in MethodNonDPCE

- **int expansionSamples**
  the expansion_samples specification in MethodNonDPCE
• **String** `expansionSampleType`  
  *incremental_lhs specification in MethodNonDPCE*

• **UShortArray** `quadratureOrder`  
  MethodNonDSC

• **unsigned short** `sparseGridLevel`  
  MethodNonDSC

• **RealVector** `sparseGridDimPref`  
  MethodNonDPCE and MethodNonDSC

• **int** `collocationPoints`  
  *the collocation_points specification in MethodNonDPCE*

• **Real** `collocationRatio`  
  *the collocation_ratio specification in MethodNonDPCE*

• **String** `collocSampleReuse`  
  *reuse_samples specification in MethodNonDPCE*

• **String** `expansionImportFile`  
  *the expansion_import_file specification in MethodNonDPCE*

• **String** `sampleType`  
  MethodNonDPCE, and MethodNonDSC.

• **String** `reliabilitySearchType`  
  MethodNonDGlobalRel *(x_gaussian_process or u_gaussian_process)*

• **String** `reliabilityIntegration`  
  MethodNonDLocalRel

• **String** `reliabilityIntegrationRefine`  
  *integration refinement selection in MethodNonDLocalRel*

• **String** `nondOptAlgorithm`  
  MethodNonDLocalRel or the interval in MethodNonDLocalIntervalEst

• **String** `distributionType`  
  *and MethodNonDGlobalRel*

• **String** `responseLevelMappingType`  
  MethodNonDLocalRel, and MethodNonDGlobalRel
• `RealVectorArray responseLevels`
  *MethodNonDPCE, MethodNonDLocalRel, and MethodNonDGlobalRel*

• `RealVectorArray probabilityLevels`
  *MethodNonDPCE, MethodNonDLocalRel, and MethodNonDGlobalRel*

• `RealVectorArray reliabilityLevels`
  *MethodNonDPCE, and MethodNonDLocalRel*

• `RealVectorArray genReliabilityLevels`
  *MethodNonDPCE, MethodNonDLocalRel, and MethodNonDGlobalRel*

• `bool allVarsFlag`
  *the all_variables specification in MethodNonDMC*

• `RealVector finalPoint`
  *the final_point.specification in MethodPSVPS*

• `RealVector stepVector`
  *the step_vector.specification in MethodPSVPS and MethodPSCPS*

• `int numSteps`
  *the num_steps.specification in MethodPSVPS*

• `IntVector stepsPerVariable`
  *the deltas_per_variable.specification in MethodPSCPS*

• `RealVector listOfPoints`
  *the list_of_points.specification in MethodPSLPS*

• `UShortArray varPartitions`
  *the partitions.specification for PStudy method in MethodPSMPS*

**Private Member Functions**

• `DataMethodRep ()`
  *constructor*

• `~DataMethodRep ()`
  *destructor*

• `void write (std::ostream &s) const`
  *write a DataInterfaceRep object to an std::ostream*

• `void read (MPIUnpackBuffer &s)`
read a DataInterfaceRep object from a packed MPI buffer

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

Private Attributes

- int referenceCount
  number of handle objects sharing this dataMethodRep

Friends

- class DataMethod
  the handle class can access attributes of the body class directly

8.28.1 Detailed Description

Body class for method specification data.

The DataMethodRep class is used to contain the data from a method keyword specification. Default values are managed in the DataMethodRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataMethodList is private (a similar approach is used with SurrogateDataPoint objects contained in Dakota::Approximation).

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

- DataMethod.H
- DataMethod.C
8.29 DataModel Class Reference

Handle class for model specification data.

Public Member Functions

- DataModel ()
  
  constructor

- DataModel (const DataModel &)
  
  copy constructor

- ~DataModel ()
  
  destructor

- DataModel & operator=(const DataModel &)
  
  assignment operator

- void write (std::ostream &s) const
  
  write a DataModel object to an std::ostream

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

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

Public Attributes

- DataModelRep * dataModelRep
  
  pointer to the body (handle-body idiom)

8.29.1 Detailed Description

Handle class for model specification data.

The DataModel class is used to provide a memory management handle for the data in DataModelRep. It is populated by IDRProblemDescDB::model_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataModel objects is maintained in ProblemDescDB::dataModelList, one for each model specification in an input file.

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

- DataModel.H
- DataModel.C
8.30 DataModelRep Class Reference

Body class for model specification data.

Public Attributes

- String idModel
  
  *the id_model specification in ModelIndControl*

- String modelType
  
  *specification in ModelIndControl*

- String variablesPointer
  
  *(from the variables_pointer specification in ModelIndControl)*

- String interfacePointer
  
  *the optional_interface_pointer specification in ModelNested*

- String responsesPointer
  
  *(from the responses_pointer specification in ModelIndControl)*

- String subMethodPointer
  
  *ModelNested*

- IntSet surrogateFnIndices
  
  *array specifying the response function set that is approximated*

- String surrogateType
  
  *polynomial, kriging, or hierarchical*

- String truthModelPointer
  
  *specification in ModelSurrH*

- String lowFidelityModelPointer
  
  *specification in ModelSurrH*

- String approxSampleReuse
  
  *ModelSurrG*

- String approxSampleReuseFile
  
  *specification in ModelSurrG*

- String approxCorrectionType
  
  *in ModelSurrG and ModelSurrH*
• short approxCorrectionOrder
  and ModelSurrH)

• bool approxGradUsageFlag
  (from the use_gradients specification in ModelSurrG)

• short polynomialOrder
  in ModelSurrG)

• RealVector krigingCorrelations
  (from the correlations specification in ModelSurrG)

• RealVector krigingConminSeed
  (from the correlations specification in ModelSurrG)

• short krigingMaxTrials
  maximum number of trials in optimization of kriging correlations

• RealVector krigingMaxCorrelations
  upper bound on kriging correlation vector

• RealVector krigingMinCorrelations
  lower bound on kriging correlation vector

• short mlsPolyOrder
  polynomial order for moving least squares approximation

• short mlsWeightFunction
  weight function for moving least squares approximation

• short rbfBases
  bases for radial basis function approximation

• short rbfMaxPts
  maximum number of points for radial basis function approximation

• short rbfMaxSubsets
  maximum number of subsets for radial basis function approximation

• short rbfMinPartition
  minimum partition for radial basis function approximation

• short marsMaxBases
  maximum number of bases for MARS approximation

• String marsInterpolation
interpolation type for MARS approximation

- short `annRandomWeight`
  random weight for artificial neural network approximation

- short `annNodes`
  number of nodes for artificial neural network approximation

- Real `annRange`
  range for artificial neural network approximation

- short `trendOrder`
  
  _gaussian_process specification in ModelSurrG_

- bool `pointSelection`
  
  flag indicating the use of point selection in the Gaussian process

- `StringArray diagMetrics`
  
  goodness of fit for a surrogate model.

- `String optionalInterfRespPointer`
  
  _optional_interface_responses_pointer specification in ModelNested_

- `StringArray primaryVarMaps`
  
  _ModelNested_

- `StringArray secondaryVarMaps`
  
  _secondary_variable_mapping specification in ModelNested_

- `RealVector primaryRespCoeffs`
  
  _specification in ModelNested_

- `RealVector secondaryRespCoeffs`
  
  _specification in ModelNested_

Private Member Functions

- `DataModelRep ()`
  
  constructor

- `~DataModelRep ()`
  
  destructor

- void `write (std::ostream &s) const`
  
  write a DataModelRep object to an std::ostream
8.30 DataModelRep Class Reference

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

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

Private Attributes

- int referenceCount
  
  number of handle objects sharing this DataModelRep

Friends

- class DataModel
  
  the handle class can access attributes of the body class directly

8.30.1 Detailed Description

Body class for model specification data.

The DataModelRep class is used to contain the data from a model keyword specification. Default values are managed in the DataModelRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataModelList is private (a similar approach is used with SurrogateDataPoint objects contained in Dakota::Approximation).

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

- DataModel.H
- DataModel.C
8.31 DataResponses Class Reference

Handle class for responses specification data.

Public Member Functions

- DataResponses ()
  * constructor

- DataResponses (const DataResponses &)
  * copy constructor

- ~DataResponses ()
  * destructor

- DataResponses & operator= (const DataResponses &)
  * assignment operator

- void write (std::ostream &s) const
  * write a DataResponses object to an std::ostream

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

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

Public Attributes

- DataResponsesRep * dataRespRep
  * pointer to the body (handle-body idiom)

8.31.1 Detailed Description

Handle class for responses specification data.

The DataResponses class is used to provide a memory management handle for the data in DataResponsesRep. It is populated by IDRProblemDescDB::responses_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataResponses objects is maintained in ProblemDescDB::dataResponsesList, one for each responses specification in an input file.

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

- DataResponses.H
- DataResponses.C
8.32 DataResponsesRep Class Reference

Body class for responses specification data.

Public Attributes

- `size_t numObjectiveFunctions`
  
  `num_objective_functions` specification in `RespFnOpt`

- `size_t numNonlinearIneqConstraints`
  
  `num_nonlinear_inequality_constraints` specification in `RespFnOpt`

- `size_t numNonlinearEqConstraints`
  
  `num_nonlinear_equality_constraints` specification in `RespFnOpt`

- `size_t numLeastSqTerms`
  
  `num_least_squares_terms` specification in `RespFnLS`

- `size_t numResponseFunctions`
  
  `num_response_functions` specification in `RespFnGen`

- `StringArray primaryRespFnScaleTypes`
  
  `the least_squares_term_scale_types` specification in `RespFnLS`

- `RealVector primaryRespFnScales`
  
  `the least_squares_term_scales` specification in `RespFnLS`

- `RealVector primaryRespFnWeights`
  
  specification in `RespFnLS`

- `String leastSqDataFile`
  
  `RespFnLS`

- `RealVector nonlinearIneqLowerBnds`
  
  `nonlinear_inequality_lower_bounds` specification in `RespFnOpt`

- `RealVector nonlinearIneqUpperBnds`
  
  `nonlinear_inequality_upper_bounds` specification in `RespFnOpt`

- `StringArray nonlinearIneqScaleTypes`
  
  `nonlinear_inequality_scale_types` specification in `RespFnOpt`

- `RealVector nonlinearIneqScales`
  
  `nonlinear_inequality_scales` specification in `RespFnOpt`
• RealVector \texttt{nonlinearEqTargets}
  \[\text{nonlinear\_equality\_targets specification in \texttt{RespFnOpt}}\]

• StringArray \texttt{nonlinearEqScaleTypes}
  \[\text{nonlinear\_equality\_scale\_types specification in \texttt{RespFnOpt}}\]

• RealVector \texttt{nonlinearEqScales}
  \[\text{nonlinear\_equality\_scales specification in \texttt{RespFnOpt}}\]

• String \texttt{gradientType}
  \[\text{mixed\_gradients specifications in \texttt{RespGrad}}\]

• String \texttt{hessianType}
  
  \[\text{RespHess}}\]

• bool \texttt{ignoreBounds}
  \[\text{is to honor bounds}\]

• bool \texttt{centralHess}
  \[\text{finite-difference Hessians; default is forward differences.}\]

• String \texttt{quasiHessianType}
  
  \[\text{and \texttt{sr1 specifications in \texttt{RespHess}}}\]

• String \texttt{methodSource}
  \[\text{method\_source specification in \texttt{RespGradNum and RespGradMixed}}\]

• String \texttt{intervalType}
  \[\text{interval\_type specification in \texttt{RespGradNum and RespGradMixed}}\]

• RealVector \texttt{fdGradStepSize}
  \[\text{specification in \texttt{RespGradNum and RespGradMixed}}\]

• RealVector \texttt{fdHessStepSize}
  
  \[\text{RespHessMixed}}\]

• IntList \texttt{idNumericalGrads}
  \[\text{specification in \texttt{RespGradMixed}}\]

• IntList \texttt{idAnalyticGrads}
  \[\text{specification in \texttt{RespGradMixed}}\]

• IntList \texttt{idNumericalHessians}
  \[\text{specification in \texttt{RespHessMixed}}\]

• IntList \texttt{idQuasiHessians}
specification in RespHessMixed)

- IntList idAnalyticHessians
  specification in RespHessMixed)

- String idResponses
  (from the id_responses specification in RespSetId)

- StringArray responseLabels
  specification in RespLabels)

Private Member Functions

- DataResponsesRep ()
  constructor

- ~DataResponsesRep ()
  destructor

- void write (std::ostream &s) const
  write a DataResponsesRep object to an std::ostream

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

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

Private Attributes

- int referenceCount
  number of handle objects sharing this dataResponsesRep

Friends

- class DataResponses
  the handle class can access attributes of the body class directly
8.32.1 Detailed Description

Body class for responses specification data.

The DataResponsesRep class is used to contain the data from a responses keyword specification. Default values are managed in the DataResponsesRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataResponsesList is private (a similar approach is used with SurrogateDataPoint objects contained in Dakota::Approximation).

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

- DataResponses.H
- DataResponses.C
8.33 DataStrategy Class Reference

Handle class for strategy specification data.

Public Member Functions

- DataStrategy ()
  constructor

- DataStrategy (const DataStrategy &)
  copy constructor

- ~DataStrategy ()
  destructor

- DataStrategy & operator= (const DataStrategy &)
  assignment operator

- void write (std::ostream &s) const
  write a DataStrategy object to an std::ostream

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

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

Public Attributes

- DataStrategyRep * dataStratRep
  pointer to the body (handle-body idiom)

8.33.1 Detailed Description

Handle class for strategy specification data.

The DataStrategy class is used to provide a memory management handle for the data in DataStrategyRep. It is populated by IDRProblemDescDB::strategy_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A single DataStrategy object is maintained in ProblemDescDB::strategySpec.

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

- DataStrategy.H
- DataStrategy.C
8.34 DataStrategyRep Class Reference

Body class for strategy specification data.

Public Attributes

- String **strategyType**
  *the strategy selection: hybrid, multi_start, pareto_set, or single_method*

- bool **graphicsFlag**
  *specification in StratIndControl*

- bool **tabularDataFlag**
  *the tabular_graphics_data specification in StratIndControl*

- String **tabularDataFile**
  *the tabular_graphics_file specification in StratIndControl*

- int **iteratorServers**
  *the iterator_servers specification in StratIndControl*

- String **iteratorScheduling**
  *iterator_static_scheduling specifications in StratIndControl*

- String **methodPointer**
  *specifications in StratSingle and StratMultiStart*

- StringArray **hybridMethodList**
  *in StratHybrid*

- String **hybridType**
  *embedded, and sequential specifications in StratHybrid*

- String **hybridGlobalMethodPointer**
  *global_method_pointer specification in StratHybrid*

- String **hybridLocalMethodPointer**
  *local_method_pointer specification in StratHybrid*

- Real **hybridLSProb**
  *local_search_probability specification in StratHybrid*

- size_t **hybridNumSolnsTrans**
  *subsequent method in the sequential hybrid optimization strategy*
Private Member Functions

- **DataStrategyRep ()**
  
  *constructor*

- **~DataStrategyRep ()**
  
  *destructor*

- **void write (std::ostream &s) const**
  
  *write a DataStrategyRep object to an std::ostream*

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

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

Private Attributes

- **int referenceCount**
  
  *number of handle objects sharing this dataStrategyRep*

Friends

- **class DataStrategy**
  
  *the handle class can access attributes of the body class directly*
8.34.1 Detailed Description

Body class for strategy specification data.

The `DataStrategyRep` class is used to contain the data from the strategy keyword specification. Default values are managed in the `DataStrategyRep` constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within `ProblemDescDB` since `ProblemDescDB::strategySpec` is private (a similar approach is used with `SurrogateDataPoint` objects contained in `Dakota::Approximation`).

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

- DataStrategy.H
- DataStrategy.C
8.35  **DataVariables Class Reference**

Handle class for variables specification data.

**Public Member Functions**

- **DataVariables ()**
  
  *constructor*

- **DataVariables (const DataVariables &)**
  
  *copy constructor*

- **~DataVariables ()**
  
  *destructor*

- **DataVariables operator= (const DataVariables &)**
  
  *assignment operator*

- **bool operator== (const DataVariables &)**
  
  *equality operator*

- **void write (std::ostream &s) const**
  
  *write a DataVariables object to an std::ostream*

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

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

- **size_t design ()**
  
  *return total number of design variables*

- **size_t aleatory_uncertain ()**
  
  *return total number of aleatory uncertain variables*

- **size_t epistemic_uncertain ()**
  
  *return total number of epistemic uncertain variables*

- **size_t uncertain ()**
  
  *return total number of uncertain variables*

- **size_t state ()**
  
  *return total number of state variables*
• size_t continuous_variables ()
  return total number of continuous variables

• size_t discrete_variables ()
  return total number of discrete variables

• size_t total_variables ()
  return total number of variables

Public Attributes

• DataVariablesRep * dataVarsRep
  pointer to the body (handle-body idiom)

8.35.1 Detailed Description

Handle class for variables specification data.

The DataVariables class is used to provide a memory management handle for the data in DataVariablesRep. It is populated by IDRProblemDescDB::variables_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataVariables objects is maintained in ProblemDescDB::dataVariablesList, one for each variables specification in an input file.

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

• DataVariables.H
• DataVariables.C
8.36 DataVariablesRep Class Reference

Body class for variables specification data.

Public Attributes

- **String idVariables**
  
  *(from the id_variables specification in VarSetId)*

- **size_t numContinuousDesVars**
  
  *(specification in VarDV)*

- **size_t numDiscreteDesRangeVars**
  
  *(from the discrete_design_range specification in VarDV)*

- **size_t numDiscreteDesSetIntVars**
  
  *(from the discrete_design_set_integer specification in VarDV)*

- **size_t numDiscreteDesSetRealVars**
  
  *(from the discrete_design_set_real specification in VarDV)*

- **size_t numNormalUncVars**
  
  *(specification in VarUV)*

- **size_t numLognormalUncVars**
  
  *(specification in VarUV)*

- **size_t numUniformUncVars**
  
  *(specification in VarUV)*

- **size_t numLoguniformUncVars**
  
  *(loguniform_uncertain specification in VarUV)*

- **size_t numTriangularUncVars**
  
  *(triangular_uncertain specification in VarUV)*

- **size_t numExponentialUncVars**
  
  *(exponential_uncertain specification in VarUV)*

- **size_t numBetaUncVars**
  
  *(specification in VarUV)*

- **size_t numGammaUncVars**
  
  *(specification in VarUV)*
- size_t numGumbelUncVars
  specification in VarUV)

- size_t numFrechetUncVars
  specification in VarUV)

- size_t numWeibullUncVars
  specification in VarUV)

- size_t numHistogramBinUncVars
  histogram_bin_uncertain specification in VarUV)

- size_t numPoissonUncVars
  poisson_uncertain specification in VarUV)

- size_t numBinomialUncVars
  binomial_uncertain specification in VarUV)

- size_t numNegBinomialUncVars
  negative_binomial_uncertain specification in VarUV)

- size_t numGeometricUncVars
  geometric_uncertain specification in VarUV

- size_t numHyperGeomUncVars
  hypergeometric_uncertain specification in VarUV))

- size_t numHistogramPtUncVars
  histogram_point_uncertain specification in VarUV)

- size_t numIntervalUncVars
  specification in VarUV)

- size_t numContinuousStateVars
  specification in VarSV)

- size_t numDiscreteStateRangeVars
  (from the discrete_state_range specification in VarDV)

- size_t numDiscreteStateSetIntVars
  (from the discrete_state_set_integer specification in VarDV)

- size_t numDiscreteStateSetRealVars
  (from the discrete_state_set_real specification in VarDV)

- RealVector continuousDesignVars
the continuous_design initial_point specification in VarDV)

- RealVector continuousDesignLowerBnds
  continuous_design lower_bounds specification in VarDV)

- RealVector continuousDesignUpperBnds
  continuous_design upper_bounds specification in VarDV)

- StringArray continuousDesignScaleTypes
  continuous_design scale_types specification in VarDV)

- RealVector continuousDesignScales
  continuous_design scales specification in VarDV)

- IntVector discreteDesignRangeVars
  specification in VarDV)

- IntVector discreteDesignRangeLowerBnds
  specification in VarDV)

- IntVector discreteDesignRangeUpperBnds
  upper_bounds specification in VarDV)

- IntVector discreteDesignSetIntVars
  specification in VarDV)

- RealVector discreteDesignSetRealVars
  specification in VarDV)

- IntSetArray discreteDesignSetInt
  discrete_design_set_integer set_values specification in VarDV)

- RealSetArray discreteDesignSetReal
  set_values specification in VarDV)

- StringArray continuousDesignLabels
  continuous_design descriptors specification in VarDV)

- StringArray discreteDesignRangeLabels
  specification in VarDV)

- StringArray discreteDesignSetIntLabels
  specification in VarDV)

- StringArray discreteDesignSetRealLabels
  specification in VarDV)
• RealVector `normalUncMeans`  
  `normalUncMeans` specification in VarUV)

• RealVector `normalUncStdDevs`  
  `normalUncStdDevs` specification in VarUV)

• RealVector `normalUncLowerBnds`  
  `normalUncLowerBnds` from the `nuv_lower_bounds` specification in VarUV)

• RealVector `normalUncUpperBnds`  
  `normalUncUpperBnds` from the `nuv_upper_bounds` specification in VarUV)

• RealVector `lognormalUncLambdas`  
  `lognormalUncLambdas` uncertain variables (from the `lnuv_lambdas` specification in VarUV)

• RealVector `lognormalUncZetas`  
  `lognormalUncZetas` uncertain variables (from the `lnuv_zetas` specification in VarUV)

• RealVector `lognormalUncMeans`  
  `lognormalUncMeans` `lnuv_means` specification in VarUV)

• RealVector `lognormalUncStdDevs`  
  `lognormalUncStdDevs` `lnuv_std_deviations` specification in VarUV)

• RealVector `lognormalUncErrFacts`  
  `lognormalUncErrFacts` `lnuv_error_factors` specification in VarUV)

• RealVector `lognormalUncLowerBnds`  
  `lognormalUncLowerBnds` `lnuv_lower_bounds` specification in VarUV)

• RealVector `lognormalUncUpperBnds`  
  `lognormalUncUpperBnds` `lnuv_upper_bounds` specification in VarUV)

• RealVector `uniformUncLowerBnds`  
  `uniformUncLowerBnds` `uuv_lower_bounds` specification in VarUV)

• RealVector `uniformUncUpperBnds`  
  `uniformUncUpperBnds` `uuv_upper_bounds` specification in VarUV)

• RealVector `loguniformUncLowerBnds`  
  `loguniformUncLowerBnds` `luuv_lower_bounds` specification in VarUV)

• RealVector `loguniformUncUpperBnds`  
  `loguniformUncUpperBnds` `luuv_upper_bounds` specification in VarUV)
• RealVector triangularUncModes
  *specification in VarUV*

• RealVector triangularUncLowerBnds
  *(from the `tuv_lower_bounds` specification in VarUV)*

• RealVector triangularUncUpperBnds
  *(from the `tuv_upper_bounds` specification in VarUV)*

• RealVector exponentialUncBetas
  *the `euv_betas` specification in VarUV*

• RealVector betaUncAlphas
  *the `buv_means` specification in VarUV*

• RealVector betaUncBetas
  *the `buv_std_deviations` specification in VarUV*

• RealVector betaUncLowerBnds
  *(from the `buv_lower_bounds` specification in VarUV)*

• RealVector betaUncUpperBnds
  *(from the `buv_upper_bounds` specification in VarUV)*

• RealVector gammaUncAlphas
  *the `gauv_alphas` specification in VarUV*

• RealVector gammaUncBetas
  *the `gauv_betas` specification in VarUV*

• RealVector gumbelUncAlphas
  *`guuv_alphas` specification in VarUV*

• RealVector gumbelUncBetas
  *the `guuv_betas` specification in VarUV*

• RealVector frechetUncAlphas
  *the `fuv_alphas` specification in VarUV*

• RealVector frechetUncBetas
  *the `fuv_betas` specification in VarUV*

• RealVector weibullUncAlphas
  *the `wuv_alphas` specification in VarUV*

• RealVector weibullUncBetas
the \texttt{wuv\_betas} specification in \texttt{VarUV})

- \textbf{RealVectorArray} \texttt{histogramUncBinPairs}  
  \textit{counts within NIDR.}

- \textbf{RealVector} \texttt{poissonUncLambdas}  
  \textit{the lambdas specification in \texttt{VarUV})}

- \textbf{RealVector} \texttt{binomialUncProbPerTrial}  
  \textit{from the prob\_per\_trial specification in \texttt{VarUV})}

- \textbf{IntVector} \texttt{binomialUncNumTrials}  
  \textit{from the num\_trials specification in \texttt{VarUV})}

- \textbf{RealVector} \texttt{negBinomialUncProbPerTrial}  
  \textit{variables from the prob\_per\_trial specification in \texttt{VarUV})}

- \textbf{IntVector} \texttt{negBinomialUncNumTrials}  
  \textit{from the num\_trials specification in \texttt{VarUV})}

- \textbf{RealVector} \texttt{geometricUncProbPerTrial}  
  \textit{variables from the prob\_per\_trial specification in \texttt{VarUV})}

- \textbf{IntVector} \texttt{hyperGeomUncTotalPop}  
  \textit{from the total\_population specification in \texttt{VarUV})}

- \textbf{IntVector} \texttt{hyperGeomUncSelectedPop}  
  \textit{from the selected\_population specification in \texttt{VarUV})}

- \textbf{IntVector} \texttt{hyperGeomUncNumDrawn}  
  \textit{variables from the num\_drawn specification in \texttt{VarUV})}

- \textbf{RealVectorArray} \texttt{histogramUncPointPairs}  
  \textit{from the histogram\_point\_uncertain specification in \texttt{VarUV})}

- \textbf{RealVectorArray} \texttt{intervalUncBasicProbs}  
  \textit{\texttt{iuv\_interval\_probs specification in \texttt{VarUV}}}

- \textbf{RealVectorArray} \texttt{intervalUncBounds}  
  \textit{\texttt{iuv\_interval\_bounds specification in \texttt{VarUV})}

- \textbf{RealSymMatrix} \texttt{uncertainCorrelations}  
  \textit{matrix) for analytic reliability methods.}

- \textbf{RealVector} \texttt{continuousStateVars}  
  \textit{the continuous\_state initial\_point specification in \texttt{VarSV})}
- RealVector `continuousStateLowerBnds
  continuous_state_lower_bounds specification in VarSV)

- RealVector `continuousStateUpperBnds
  continuous_state_upper_bounds specification in VarSV)

- IntVector `discreteStateRangeVars
  specification in VarSV)

- IntVector `discreteStateRangeLowerBnds
  specification in VarSV)

- IntVector `discreteStateRangeUpperBnds
  upper_bounds specification in VarSV)

- IntVector `discreteStateSetIntVars
  specification in VarSV)

- RealVector `discreteStateSetRealVars
  specification in VarSV)

- IntSetArray `discreteStateSetInt
  discrete_state_set_integer set_values specification in VarSV)

- RealSetArray `discreteStateSetReal
  set_values specification in VarSV)

- StringArray `continuousStateLabels
  continuous_state_descriptors specification in VarSV)

- StringArray `discreteStateRangeLabels
  specification in VarSV)

- StringArray `discreteStateSetIntLabels
  specification in VarSV)

- StringArray `discreteStateSetRealLabels
  specification in VarSV)

- IntVector `discreteDesignSetIntLowerBnds
  discrete design integer set lower bounds inferred from set values

- IntVector `discreteDesignSetIntUpperBnds
  discrete design integer set upper bounds inferred from set values
• RealVector `discreteDesignSetRealLowerBnds`
  discrete design real set lower bounds inferred from set values

• RealVector `discreteDesignSetRealUpperBnds`
  discrete design real set upper bounds inferred from set values

• RealVector `continuousAleatoryUncVars`
  array of values for all continuous aleatory uncertain variables

• RealVector `continuousAleatoryUncLowerBnds`
  for gamma, gumbel, frechet, weibull and histogram bin specifications)

• RealVector `continuousAleatoryUncUpperBnds`
  for gamma, gumbel, frechet, weibull and histogram bin specifications)

• StringArray `continuousAleatoryUncLabels`
  specifications in VarUV)

• IntVector `discreteIntAleatoryUncVars`
  array of values for all discrete integer aleatory uncertain variables

• IntVector `discreteIntAleatoryUncLowerBnds`
  uncertain variables

• IntVector `discreteIntAleatoryUncUpperBnds`
  uncertain variables

• StringArray `discreteIntAleatoryUncLabels`
  labels for all discrete integer aleatory uncertain variables

• RealVector `discreteRealAleatoryUncVars`
  array of values for all discrete real aleatory uncertain variables

• RealVector `discreteRealAleatoryUncLowerBnds`
  uncertain variables

• RealVector `discreteRealAleatoryUncUpperBnds`
  uncertain variables

• StringArray `discreteRealAleatoryUncLabels`
  labels for all discrete real aleatory uncertain variables

• RealVector `continuousEpistemicUncVars`
  array of values for all continuous epistemic uncertain variables

• RealVector `continuousEpistemicUncLowerBnds`
distribution lower bounds for all continuous epistemic uncertain variables

- **RealVector** continuousEpistemicUncUpperBnds
  distribution upper bounds for all continuous epistemic uncertain variables

- **StringArray** continuousEpistemicUncLabels
  labels for all continuous epistemic uncertain variables

- **IntVector** discreteStateSetIntLowerBnds
  discrete state integer set lower bounds inferred from set values

- **IntVector** discreteStateSetIntUpperBnds
  discrete state integer set upper bounds inferred from set values

- **RealVector** discreteStateSetRealLowerBnds
  discrete state real set lower bounds inferred from set values

- **RealVector** discreteStateSetRealUpperBnds
  discrete state real set upper bounds inferred from set values

**Private Member Functions**

- **DataVariablesRep ()**
  default constructor

- **~DataVariablesRep ()**
  destructor

- void **write** (std::ostream &s) const
  write a DataVariablesRep object to an std::ostream

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

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

**Private Attributes**

- **int** referenceCount
  number of handle objects sharing dataVarsRep

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
Friends

- class DataVariables

  the handle class can access attributes of the body class directly

8.36.1 Detailed Description

Body class for variables specification data.

The DataVariablesRep class is used to contain the data from a variables keyword specification. Default values are managed in the DataVariablesRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataVariablesList is private (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
8.37  DDACEDesignCompExp Class Reference

Wrapper class for the DDACE design of experiments library.

Inheritance diagram for DDACEDesignCompExp::

```
  Iterator
   |    
  Analyzer
   |    
PStudyDACE
   |    
DDACEDesignCompExp
```

Public Member Functions

- **DDACEDesignCompExp (Model &model)**
  - primary constructor for building a standard DACE iterator

- **DDACEDesignCompExp (Model &model, int samples, int symbols, int seed, const String &sampling_method)**
  - alternate constructor used for building approximations

- **~DDACEDesignCompExp ()**
  - destructor

- **void pre_run ()**
  - pre-run portion of run_iterator (optional)

- **void extract_trends ()**
  - Redefines the run_iterator virtual function for the PStudy/DACE branch.

- **void derived_post_run ()**
  - portions of post_run specific to derived iterators

- **void sampling_reset (int min_samples, int rec_samples, bool all_data_flag, bool stats_flag)**
  - reset sampling iterator

- **const String & sampling_scheme () const**
  - return sampling name

- **void vary_pattern (bool pattern_flag)**
  - sets varyPattern in derived classes that support it
• void `get_parameter_sets` (const `Model` &model)
  Returns one block of samples \((n_{\text{dim}} \ast n_{\text{num} \_\text{samples}})\).

**Private Member Functions**

• void `compute_main_effects` ()
  builds a `DDaceMainEffects::OneWayANOVA` if `mainEffectsFlag` is set

• void `resolve_samples_symbols` ()
  number of symbols from input.

**Private Attributes**

• `String` `daceMethod`
  *oas, lhs, oa\_lhs, random, box\_behnken, central\_composite, or grid*

• `int` `samplesSpec`
  initial specification of number of samples

• `int` `symbolsSpec`
  initial specification of number of symbols

• `int` `numSamples`
  current number of samples to be evaluated

• `int` `numSymbols`
  (inversely related to number of replications)

• `const int` `seedSpec`
  (allows repeatable results)

• `int` `randomSeed`
  current seed for the random number generator

• `bool` `allDataFlag`
  `Iterator::all\_variables()` and `Iterator::all\_responses()`.

• `size_t` `numDACERuns`
  counter for number of `run()` executions for this object

• `bool` `varyPattern`
  multiple executions are repeatable but not correlated.
8.37 DDACEDesignCompExp Class Reference

- bool varBasedDecompFlag
  flag which specifies variance based decomposition

- bool mainEffectsFlag
  flag which specifies main effects

- std::vector< std::vector< int > > symbolMapping
  mapping of symbols for main effects calculations

8.37.1 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.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 DDACEDesignCompExp (Model & model)

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

8.37.2.2 DDACEDesignCompExp (Model & model, int samples, int symbols, int seed, const String & sampling_method)

alternate constructor used for building approximations
This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.

8.37.3 Member Function Documentation

8.37.3.1 void pre_run () [virtual]

pre-run portion of run_iterator (optional)
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely contained in the derived run function
Reimplemented from Iterator.
8.37.3.2 void derived_post_run () [virtual]

portions of post_run specific to derived iterators

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of post_run(). Redefinition by derived classes is optional.

Reimplemented from Iterator.

8.37.3.3 void resolve_samples_symbols () [private]

number of symbols from input.

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

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

- DDACEDesignCompExp.H
- DDACEDesignCompExp.C
and testers using direct procedure calls.

Inheritance diagram for DirectApplicInterface::

```
  Interface
    ApplicationInterface
      DirectApplicInterface
          ParallelDirectApplicInterface  SerialDirectApplicInterface
```

### Public Member Functions

- **DirectApplicInterface** (const `ProblemDescDB` &`problem_db`)
  
  constructor

- **~DirectApplicInterface** ()
  
  destructor

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

- **void derived_map_asynch** (const `ParamResponsePair` &`pair`)
  
  asynchronous evaluation that is specific to a derived class.

- **void derived_synch** (`PRPQueue` &`prp_queue`)
  
  classes. This version waits for at least one completion.

- **void derived_synch_nowait** (`PRPQueue` &`prp_queue`)
  
  any completions if none are immediately available.

- **int derived_synchronous_local_analysis** (const int &`analysis_id`)

- **const `StringArray` & analysis_drivers** () const
  
  retrieve the analysis drivers specification for application interfaces

### Protected Member Functions

- **virtual int derived_map_if** (const `Dakota::String` &`if_name`)
  
  execute the input filter portion of a direct evaluation invocation
• virtual int derived_map_ac (const Dakota::String &ac_name)
  execute an analysis code portion of a direct evaluation invocation

• virtual int derived_map_of (const Dakota::String &of_name)
  execute the output filter portion of a direct evaluation invocation

• void set_local_data (const Variables &vars, const ActiveSet &set, const Response &response)
  variable and response attributes

• void overlay_response (Response &response)
  response contributions from multiple analyses using MPI_Reduce

Protected Attributes

• String iFilterName
  name of the direct function input filter

• String oFilterName
  name of the direct function output filter

• driver_t iFilterType
  enum type of the direct function input filter

• driver_t oFilterType
  enum type of the direct function output filter

• bool gradFlag
  signals use of fnGrads in direct simulator functions

• bool hessFlag
  signals use of fnHessians in direct simulator functions

• size_t numFns
  number of functions in fnVals

• size_t numVars
  total number of continuous and discrete variables

• size_t numACV
  total number of continuous variables

• size_t numADIV
  total number of discrete integer variables
- `size_t numADRv`  
  total number of discrete real variables

- `size_t numDerivVars`  
  number of active derivative variables

- `unsigned short localDataView`  
  see enum local_data_t

- `RealVector xC`  
  continuous variables used within direct simulator fns

- `IntVector xDI`  
  discrete int variables used within direct simulator fns

- `RealVector xDR`  
  discrete real variables used within direct simulator fns

- `StringMultiArray xCLabels`  
  continuous variable labels

- `StringMultiArray xDILabels`  
  discrete integer variable labels

- `StringMultiArray xDRLabels`  
  discrete real variable labels

- `std::map< String, var_t > varTypeMap`  
  map from variable label to enum

- `std::map< String, driver_t > driverTypeMap`  
  map from driver name to enum

- `std::map< var_t, Real > xCM`  
  map from var_t enum to continuous value

- `std::map< var_t, int > xDIM`  
  map from var_t enum to discrete int value

- `std::map< var_t, Real > xDRM`  
  map from var_t enum to discrete real value

- `std::vector< var_t > varTypeDVV`  
  var_t enumerations corresponding to DVV components

- `std::vector< var_t > xCMLabels`
var_t enumerations corresponding to continuous variable labels

- std::vector< var_t > xDIMLabels

var_t enumerations corresponding to discrete integer variable labels

- std::vector< var_t > xDRMLabels

var_t enumerations corresponding to discrete real variable labels

- ShortArray directFnASV
  class scope active set vector

- UIntArray directFnDVV
  class scope derivative variables vector

- RealVector fnVals
  response fn values within direct simulator fns

- RealMatrix fnGrads
  response fn gradients w/ direct simulator fns

- RealSymMatrixArray fnHessians
  response fn Hessians within direct fns

- StringArray analysisDrivers
  analysis_drivers interface specification)

- Array< driver_t > analysisDriverTypes
  conversion of analysisDrivers to driver_t

- size_t analysisDriverIndex
  the index of the active analysis driver within analysisDrivers

- String2DArray analysisComponents
  (from the analysis_components interface specification)

- engine * matlabEngine
  pointer to the MATLAB engine used for direct evaluations

Private Member Functions

- int cantilever ()
  the cantilever UQ/OUU test function

- int cyl_head ()
  the cylinder head constrained optimization test fn
- int multimodal ()
  
  multimodal UQ test function

- int rosenbrock ()
  
  the Rosenbrock optimization and least squares test fn

- int generalized_rosenbrock ()
  
  n-dimensional Rosenbrock (Schittkowski)

- int extended_rosenbrock ()
  
  n-dimensional Rosenbrock (Nocedal/Wright)

- int log_ratio ()
  
  the log_ratio UQ test function

- int short_column ()
  
  the short_column UQ/OUU test function

- int steel_column_cost ()
  
  the steel_column_cost UQ/OUU test function

- int steel_column_perf ()
  
  the short_column_perf UQ/OUU test function

- int sobol_rational ()
  
  Sobol SA rational test function.

- int sobol_g_function ()
  
  Sobol SA discontinuous test function.

- int sobol_jishigami ()
  
  Sobol SA transcendental test function.

- int text_book ()
  
  the text_book constrained optimization test function

- int text_book1 ()
  
  portion of text_book() evaluating the objective fn

- int text_book2 ()
  
  portion of text_book() evaluating constraint 1

- int text_book3 ()
  
  portion of text_book() evaluating constraint 2
• int text_book_ouu ()
  
  the text_book_ouu OUU test function

• int salinas ()
  
  direct interface to the SALINAS structural dynamics code

• int mc_api_run ()
  
  direct interface to ModelCenter via API, HKIM 4/3/03

• int matlab_engine_run ()
  
  direct interface to Matlab via API, BMA 11/28/05

• int matlab_field_prep (mxArray *dakota_matlab, const char *field_name)
  
  add if necessary; free structure memory in preparation for new alloc

• int python_run ()
  
  direct interface to Python via API, BMA 07/02/07

• template<class ArrayT> bool python_convert_int (const ArrayT &src, PyObject **dst)
  
  convert arrays of integer types to Python list or numpy array

• bool python_convert (const RealVector &src, PyObject **dst)
  
  convert RealVector to Python list or numpy array

• bool python_convert (const RealVector &c_src, const IntVector &di_src, const RealVector &dr_src, PyObject **dst)
  
  or numpy double array

• bool python_convert (const StringMultiArray &src, PyObject **dst)
  
  convert labels

• bool python_convert (const StringMultiArray &c_src, const StringMultiArray &di_src, const StringMultiArray &dr_src, PyObject **dst)
  
  convert all labels to single list

• bool python_convert (PyObject *pyv, RealVector &rv, const int &dim)
  
  RealVector (for fns).

• bool python_convert (PyObject *pyv, double *rv, const int &dim)
  
  double[], for use as helper in converting gradients

• bool python_convert (PyObject *pym, RealMatrix &rm)
  
  to RealMatrix (for gradients)

• bool python_convert (PyObject *pym, RealSymMatrix &rm)
  
  to RealMatrix (used as helper in Hessian conversion)
8.38 DirectApplicInterface Class Reference

- bool python_convert (PyObject *pyma, RealSymMatrixArray &rma)

  [numpy array of double] to RealSymMatrixArray (for Hessians)

Private Attributes

- bool userNumpyFlag

  whether the user requested numpy data structures

8.38.1 Detailed Description

and testers using direct procedure calls.

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

8.38.2 Member Function Documentation

8.38.2.1 int derived_synchronous_local_analysis (const int & analysis_id) [inline, virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().
Reimplemented from ApplicationInterface.

8.38.2.2 int derived_map_ac (const Dakota::String & ac_name) [protected, virtual]

execute an analysis code portion of a direct evaluation invocation

When a direct analysis/filter is a member function, the (vars,set,response) data does not need to be passed through the API. If, however, non-member analysis/filter functions are added, then pass (vars,set,response) through to the non-member fns:

```c++
// API declaration
int sim(const Variables& vars, const ActiveSet& set, Response& response);
// use of API within derived_map_ac()
if (ac_name == "sim")
  fail_code = sim(directFnVars, directFnActSet, directFnResponse);
```

Reimplemented in ParallelDirectApplicInterface, and SerialDirectApplicInterface.

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

- DirectApplicInterface.H
- DirectApplicInterface.C
8.39  DOTOptimizer Class Reference

Wrapper class for the DOT optimization library.

Inheritance diagram for DOTOptimizer::

```
DOTOptimizer
    ├── Optimizer
    │    ├── Minimizer
    │    │    └── Iterator

DOTOptimizer
    ├── Optimizer
    │    ├── Minimizer
    │    │    └── Iterator
```

Public Member Functions

- **DOTOptimizer (Model &model)**
  
  *standard constructor*

- **DOTOptimizer (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor*

- **~DOTOptimizer ()**
  
  *destructor*

- **void find_optimum ()**
  
  *Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- **void derived_initialize_run ()**
  
  *performs run-time set up*

Private Member Functions

- **void initialize ()**
  
  *Shared constructor code.*

- **void allocate_workspace ()**
  
  *Allocates workspace for the optimizer.*
• void allocate_constraints ()
   
   Allocates constraint mappings.

Private Attributes

• int dotInfo
   
   INFO from DOT manual.

• int dotFDsInfo
   
   internal DOT parameter NGOTOZ

• int dotMethod
   
   METHOD from DOT manual.

• int printControl
   
   IPRINT from DOT manual (controls output verbosity).

• int optimizationType
   
   MINMAX from DOT manual (minimize or maximize).

• RealArray realCntlParmArray
   
   RPRM from DOT manual.

• IntArray intCntlParmArray
   
   IPRM from DOT manual.

• RealVector designVars
   
   array of design variable values passed to DOT

• Real objFnValue
   
   value of the objective function passed to DOT

• RealVector constraintValues
   
   array of nonlinear constraint values passed to DOT

• int realWorkSpaceSize
   
   size of realWorkSpace

• int intWorkSpaceSize
   
   size of intWorkSpace

• RealArray realWorkSpace
   
   real work space for DOT
8.39.1 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.

8.39.2 Member Data Documentation

8.39.2.1 int dotInfo [private]

INFO from DOT manual.

Information requested by DOT: 0=optimization complete, 1=get values, 2=get gradients
8.39.2.2 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).

8.39.2.3 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.

8.39.2.4 int printControl [private]

IPRINT from DOT manual (controls output verbosity).
Values range from 0 (least output) to 7 (most output).

8.39.2.5 int optimizationType [private]

MINMAX from DOT manual (minimize or maximize).
Values of 0 or -1 (minimize) or 1 (maximize).

8.39.2.6 RealArray realCntlParmArray [private]

RPRM from DOT manual.
Array of real control parameters.

8.39.2.7 IntArray intCntlParmArray [private]

IPRM from DOT manual.
Array of integer control parameters.

8.39.2.8 RealVector constraintValues [private]

array of nonlinear constraint values passed to DOT
This array must be of nonzero length and must contain only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities and equalities).

8.39.2.9 SizetArray constraintMappingIndices [private]

Response constraints used in computing the DOT constraints.
The length of the container corresponds to the number of DOT constraints, and each entry in the container points to the corresponding DAKOTA constraint.

8.39.2.10 RealArray constraintMappingMultipliers [private]

The length of the container corresponds to the number of DOT constraints, and each entry in the container stores a multiplier for the DAKOTA constraint identified with constraintMappingIndices. These multipliers are currently +1 or -1.

8.39.2.11 RealArray constraintMappingOffsets [private]

The length of the container corresponds to the number of DOT constraints, and each entry in the container stores an offset for the DAKOTA constraint identified with constraintMappingIndices. These offsets involve inequality bounds or equality targets, since DOT assumes constraint allowables = 0.

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

- DOTOptimizer.H
- DOTOptimizer.C
8.40  EffGlobalMinimizer Class Reference

Implementation of Efficient Global Optimization/Least Squares algorithms.

Inheritance diagram for EffGlobalMinimizer:

```
EffGlobalMinimizer
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SurroBasedMinimizer</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Minimizer</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Iterator</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **EffGlobalMinimizer (Model &model)**
  - *standard constructor*

- **~EffGlobalMinimizer ()**
  - *alternate constructor for instantiations “on the fly” destructor*

- **void minimize_surrogates ()**
  - *approach. Redefines the Iterator::run() virtual function.*

Private Member Functions

- **void minimize_surrogates_on_model ()**
  - *called by minimize_surrogates for setUpType == “model”*

- **void get_best_sample ()**
  - *improvement function*

- **Real expected_improvement (const RealVector &means, const RealVector &variances)**
  - *expected improvement function for the GP*

- **RealVector expected_violation (const RealVector &means, const RealVector &variances)**
  - *expected violation function for the constraint functions*

- **void update_penalty ()**
  - *initialize and update the penaltyParameter*
• Real rel_change_c_star (const RealVector &curr_c_star, const RealVector &prev_c_star)
  Computes relative change between successive c_stars using Euclidean norm.

Static Private Member Functions

• static void EIF_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  Expected Improvement (EIF) problem formulation for PMA.

Private Attributes

• String setUpType
  (user-supplied functions mode for "on the fly" instantiations).

• Model fHatModel
  GP model of response, one approximation per response function.

• Model eifModel
  max(EIF) sub-problem

• Real meritFnStar
  minimum penalized response from among true function evaluations

• RealVector truthFnStar
  true function values corresponding to the minimum penalized response

• RealVector varStar
  point that corresponds to the optimal value meritFnStar

Static Private Attributes

• static EffGlobalMinimizer * effGlobalInstance
  functions in order to avoid the need for static data

8.40.1 Detailed Description

Implementation of Efficient Global Optimization/Least Squares algorithms.

The EffGlobalMinimizer class provides an implementation of the Efficient Global Optimization algorithm developed by Jones, Schonlau, & Welch as well as adaptation of the concept to nonlinear least squares.
8.40.2 Constructor & Destructor Documentation

8.40.2.1 \texttt{EffGlobalMinimizer} ()

alternate constructor for instantiations "on the fly" destructor

This is an alternate constructor for instantiations on the fly using a \texttt{Model} but no \texttt{ProblemDescDB}.

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

- \texttt{EffGlobalMinimizer.H}
- \texttt{EffGlobalMinimizer.C}
8.41 EmbeddedHybridStrategy Class Reference

search methods.

Inheritance diagram for EmbeddedHybridStrategy:

```
Strategy

HybridStrategy

EmbeddedHybridStrategy
```

Public Member Functions

- `EmbeddedHybridStrategy (ProblemDescDB &problem_db)`
  constructor

- `~EmbeddedHybridStrategy ()`
  destructor

Protected Member Functions

- `void run_strategy ()`
  *iterators on different models of varying fidelity*

- `const Variables & variables_results () const`
  *return the final solution from selectedIterators (variables)*

- `const Response & response_results () const`
  *return the final solution from selectedIterators (response)*

Private Attributes

- `Real localSearchProb`
  *phases of the global minimization for coupled hybrids*

8.41.1 Detailed Description

search methods.
This strategy uses multiple methods in close coordination, generally using a local search minimizer repeatedly within a global minimizer (the local search minimizer refines candidate minima which are fed back to the global minimizer).

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

- EmbeddedHybridStrategy.H
- EmbeddedHybridStrategy.C
8.42 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.

8.42.1 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
8.43 ForkAnalysisCode Class Reference

simulations using forks.

Inheritance diagram for ForkAnalysisCode:

```
  AnalysisCode
   |       |
  ForkAnalysisCode
```

Public Member Functions

- **ForkAnalysisCode** (const ProblemDescDB &problem_db)  
  constructor

- **~ForkAnalysisCode** ()  
  destructor

- **pid_t fork_program** (const bool block_flag)  
  for completion using waitpid() if block_flag is true

- **void check_status** (const int status)  
  error code was returned

- **void ifilter_argument_list** ()  
  set argList for execution of the input filter

- **void ofilter_argument_list** ()  
  set argList for execution of the output filter

- **void driver_argument_list** (const int analysis_id)  
  set argList for execution of the specified analysis driver

Private Attributes

- **StringArray argList**  
  These are converted to an array of const char*s in fork_program().
8.43.1 Detailed Description

simulations using forks.

ForkAnalysisCode creates a copy of the parent DAKOTA process using fork()/vfork() and then replaces the copy with a simulation process using execvp(). The parent process can then use waitpid() to wait on completion of the simulation process.

8.43.2 Member Function Documentation

8.43.2.1 void check_status (const int status)

error code was returned

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

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

- ForkAnalysisCode.H
- ForkAnalysisCode.C
8.44 ForkApplicInterface Class Reference

using forks.

Inheritance diagram for ForkApplicInterface:

```
  Interface
   |
   v
ApplicationInterface
   |
ForkApplicInterface
```

Public Member Functions

- **ForkApplicInterface** (const ProblemDescDB &problem_db)
  constructor

- **~ForkApplicInterface** ()
  destructor

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

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

- void **derived_synch** (PRPQueue &prp_queue)
  classes. This version waits for at least one completion.

- void **derived_synch_nowait** (PRPQueue &prp_queue)
  any completions if none are immediately available.

- int **derived_synchronous_local_analysis** (const int &analysis_id)

- const **StringArray &analysis_drivers** () const
  retrieve the analysis drivers specification for application interfaces

- const **AnalysisCode *analysis_code** () const
  return AnalysisCode::fileNameMap when defined for derived Interface class

Private Member Functions

- void **derived_synch_kernel** (PRPQueue &prp_queue, const pid_t pid)
derived_synch_nowait()  

- **pid_t fork_application (const bool block_flag)**  
  filter, analysis programs, and output filter  

- **void asynchronous_local_analyses (const int &start, const int &end, const int &step)**  
  execute analyses asynchronously on the local processor  

- **void synchronous_local_analyses (const int &start, const int &end, const int &step)**  
  execute analyses synchronously on the local processor  

- **void serve_analyses_asynch ()**  
  serve the analysis scheduler and execute analysis jobs asynchronously  

### Private Attributes  

- **ForkAnalysisCode forkSimulator**  
  individual programs and checking fork exit status  

- **std::map< pid_t, int > processIdMap**  
  asynchronous evaluations  

### 8.44.1 Detailed Description  

using forks.  

ForkApplicInterface uses a ForkAnalysisCode object for performing simulation invocations.  

### 8.44.2 Member Function Documentation  

#### 8.44.2.1 int derived_synchronous_local_analysis (const int & analysis_id) [inline, virtual]  

This code provides the derived function used by ApplicationInterface::serve_analyses_synch() as well as a convenience function for ForkApplicInterface::synchronous_local_analyses() below.  

Reimplemented from ApplicationInterface.  

#### 8.44.2.2 pid_t fork_application (const bool block_flag) [private]  

filter, analysis programs, and output filter  

Manage the input filter, 1 or more analysis programs, and the output filter in blocking or nonblocking mode as governed by block_flag. In the case of a single analysis and no filters, a single fork is performed, while in other cases, an initial fork is reforked multiple times. Called from derived_map() with block_flag == BLOCK and from derived_map_asynch() with block_flag == FALL_THROUGH. Uses ForkAnalysisCode::fork_program() to spawn individual program components within the function evaluation.
8.44 ForkApplicInterface Class Reference

8.44.2.3  void asynchronous_local_analyses (const int & start, const int & end, const int & step)  
          [private]

execute analyses asynchronously on the local processor
Schedule analyses asynchronously on the local processor using a self-scheduling approach (start to end in step increments). Concurrency is limited by asynchLocalAnalysisConcurrency. Modeled after ApplicationInterface::asynchronous_local_evaluations(). NOTE: This function should be elevated to ApplicationInterface if and when another derived interface class supports asynchronous local analyses.

8.44.2.4  void synchronous_local_analyses (const int & start, const int & end, const int & step)  
           [inline, private]

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

8.44.2.5  void serve_analyses_asynch ()  [private]

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

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

- ForkApplicInterface.H
- ForkApplicInterface.C
8.45  FSUDesignCompExp Class Reference

Wrapper class for the FSUDace QMC/CVT library.

Inheritance diagram for FSUDesignCompExp:

```
          Iterator
           |
           Analyzer
           |
         PStudyDACE
           |
      FSUDesignCompExp
```

Public Member Functions

- **FSUDesignCompExp** Model &model
  
  primary constructor for building a standard DACE iterator

- **FSUDesignCompExp** Model &model, int samples, int seed, const String &sampling_method
  
  alternate constructor for building a DACE iterator on-the-fly

- **~FSUDesignCompExp**
  
  destructor

- **void pre_run**
  
  pre-run portion of run_iterator (optional)

- **void extract_trends**
  
  Redefines the run_iterator virtual function for the PStudy/DACE branch.

- **void derived_post_run**
  
  portions of post_run specific to derived iterators

- **void sampling_reset** int min_samples, int rec_samples, bool all_data_flag, bool stats_flag
  
  reset sampling iterator

- **const String & sampling_scheme** () const
  
  return sampling name

- **void vary_pattern** bool pattern_flag
  
  sets varyPattern in derived classes that support it
void get_parameter_sets (const Model &model)

Returns one block of samples \((ndim \times num\_samples)\).

### Private Member Functions

- void enforce_input_rules ()

  enforce sanity checks/modifications for the user input specification

### Private Attributes

- int samplesSpec
  initial specification of number of samples

- int numSamples
  current number of samples to be evaluated

- bool allDataFlag
  \textit{Iterator::all\_variables()} and \textit{Iterator::all\_responses()}.

- size\_t numDACERuns
  counter for number of \textit{run()} executions for this object

- bool latinizeFlag
  flag which specifies latinization of QMC or CVT sample sets

- bool varBasedDecompFlag
  sensitivity analysis metrics

- IntVector sequenceStart
  variable sampled. Default is 0 0 0 (e.g. for three random variables).

- IntVector sequenceLeap
  generated. Default is 1 1 1 (e.g. for three random vars.)

- IntVector primeBase
  generated. Default is 2 3 5 (e.g., for three random vars.)

- int seedSpec
  \textit{(allows repeatable results)}

- int randomSeed
  current seed for the random number generator
• bool varyPattern
  multiple executions are repeatable but not identical.

• int numCVTTrials
  specifies the number of sample points taken at internal CVT iteration

• int trialType
  halton (1), uniform (0), or random (-1). Default is random.

8.45.1 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 Tessellation (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.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 FSUDesignCompExp (Model & model)

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

8.45.2.2 FSUDesignCompExp (Model & model, int samples, int seed, const String & sampling_method)

alternate constructor for building a DACE iterator on-the-fly
This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.

8.45.3 Member Function Documentation

8.45.3.1 void pre_run () [virtual]

pre-run portion of run_iterator (optional)
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely contained in the derived run function
Reimplemented from Iterator.

8.45.3.2 void derived_post_run () [virtual]

portions of post_run specific to derived iterators
Iterator supports a construct/initialize-run/run/finalize-run/destroy progression. This function is the virtual derived class portion of post_run(). Redefinition by derived classes is optional.
Redefinition from Iterator.

8.45.3.3 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
8.46 FunctionCompare Class Template Reference

Public Member Functions

- **FunctionCompare** (bool (*)(func)(const T &, void *), void *)
  Constructor that defines the pointer to function and search value.

- **bool operator() (T t) const**
  The operator() must be defined. Calls the function test_fn.

Private Attributes

- **bool (* test_fn ) (const T &, void *)**
  Pointer to test function.

- **void * search_val**
  Holds the value to search for.

8.46.1 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
8.47 GaussProcApproximation Class Reference

Derived approximation class for Gaussian Process implementation.

Inheritance diagram for GaussProcApproximation::

```
Approximation
  GaussProcApproximation
```

Public Member Functions

- **GaussProcApproximation()**
  *default constructor*

- **GaussProcApproximation(const ProblemDescDB &problem_db, const size_t &num_acv)**
  *standard constructor*

- **~GaussProcApproximation()**
  *destructor*

Protected Member Functions

- **int min_coefficients() const**
  *build the derived class approximation type in numVars dimensions*

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

- **void find_coefficients()**
  *find the covariance parameters governing the Gaussian process response*

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

- **const Real & get_variance(const RealVector &x)**
  *retrieve the variance of the predicted value for a given parameter set x*

- **const RealVector & get_gradient(const RealVector &x)**
  *for a given parameter set x*
**Private Member Functions**

- **void** `GPmodel_build()`  
  *Function to compute hyperparameters governing the GP.*

- **void** `GPmodel_apply(const RealVector &new_x, bool variance_flag, bool gradients_flag)`  
  *Function returns a response value using the GP surface.*

- **void** `normalize_training_data()`  
  *Normalizes the initial inputs upon which the GP surface is based.*

- **void** `get_trend()`  
  *linear, if order=2, trend is quadratic.*

- **void** `get_beta_coefficients()`  
  *Gets the beta coefficients for the calculation of the mean of the GP.*

- **int** `get_cholesky_factor()`  
  *error checking*

- **void** `get_process_variance()`  
  *the correlation lengthscales*

- **void** `get_cov_matrix()`  
  *calculates the covariance matrix for a given set of input points*

- **void** `get_cov_vector()`  
  *set of inputs upon which the GP is based*

- **void** `optimize_theta_global()`  
  *parameters using NCSUDirect*

- **void** `optimize_theta_multipoint()`  
  *parameters using a gradient-based solver and multiple starting points*

- **void** `predict(bool variance_flag, bool gradients_flag)`  
  *Calculates the predicted new response value for x in normalized space.*

- **Real** `calc_nll()`  
  *matrix*

- **void** `calc_grad_nll()`  
  *to the correlation lengthscales, theta*

- **void** `get_grad_cov_vector()`  
  *to each component of x.*
- **void run_point_selection ()**
  
  estimate the necessary parameters

- **void initialize_point_selection ()**
  
  initial subset of the training points

- **void pointsel_get_errors (RealArray &delta)**
  
  training points and find the errors

- **int addpoint (int, IntArray &added_index)**
  
  Adds a point to the effective training set. Returns 1 on success.

- **int pointsel_add_sel (const RealArray &delta)**
  
  them

- **Real maxval (const RealArray &) const**
  
  Return the maximum value of the elements in a vector.

- **void pointsel_write_points ()**
  
  Writes out the training set before and after point selection.

- **void likelihood_2d_grid_eval ()**
  
  likelihood on a grid

- **void writex (char [])**
  
  specified file

- **void writeCovMat (char [])**
  
  Writes out the covariance matrix to a specified file.

### Static Private Member Functions

- **static void negloglik (int mode, int n, const NEWMAT::ColumnVector &X, NEWMAT::Real &fx, NEWMAT::ColumnVector &grad_x, int &result_mode)**

  by minimizing the negative log likelihood

- **static void constraint_eval (int mode, int n, const NEWMAT::ColumnVector &X, NEWMAT::ColumnVector &g, NEWMAT::Matrix &gradC, int &result_mode)**

  this function is empty: it is an unconstrained optimization.

- **static double negloglikNCSU (const RealVector &x)**

  function used by NCSU/Optimizer to optimize negloglik objective
Private Attributes

- RealMatrix $\text{trainPoints}$
  
  *used to create the Gaussian process*

- RealMatrix $\text{trainValues}$
  
  *An array of response values; one response value per sample site.*

- RealVector $\text{trainMeans}$
  
  *The mean of the input columns of trainPoints.*

- RealVector $\text{trainStdvs}$
  
  *The standard deviation of the input columns of trainPoints.*

- RealMatrix $\text{normTrainPoints}$
  
  *Current working set of normalized points upon which the GP is based.*

- RealMatrix $\text{trendFunction}$
  
  *Matrix to hold the trend function*

- RealMatrix $\text{betaCoeffs}$
  
  *Matrix to hold the beta coefficients for the trend function*

- RealSymMatrix $\text{covMatrix}$
  
  *Between points $X_i$ and $X_j$ in the initial set of samples*

- RealMatrix $\text{covVector}$
  
  *Between a new point $X$ and point $X_j$ from the initial set of samples*

- RealMatrix $\text{approxPoint}$
  
  *Single point, but it could be generalized to be a vector of points*

- RealMatrix $\text{gradNegLogLikTheta}$
  
  *With respect to the theta correlation terms*

- RealSpdSolver $\text{covSlvr}$
  
  *The covariance matrix*

- RealMatrix $\text{gradCovVector}$
  
  *With respect to a particular component of $X$*

- RealMatrix $\text{normTrainPointsAll}$
  
  *Set of all original samples available.*

- RealMatrix $\text{trainValuesAll}$
  
  *All original samples available.*
8.47 GaussProcApproximation Class Reference

- **RealMatrix** `trendFunctionAll`
  
  *Trend function values corresponding to all original samples.*

- **size_t** `numObs`
  
  *The number of observations on which the GP surface is built.*

- **size_t** `numObsAll`
  
  *The original number of observations.*

- **short** `trendOrder`
  
  *linear, if order = 2, trend is quadratic.*

- **RealVector** `thetaParams`
  
  *same point. sige is the underlying process error.*

- **Real** `procVar`
  
  *The process variance, the multiplier of the correlation matrix.*

- **IntArray** `pointsAddedIndex`
  
  *all points which have been added*

- **int** `cholFlag`
  
  *A global indicator for success of the Cholesky factorization.*

- **bool** `usePointSelection`
  
  *a flag to indicate the use of point selection*

### Static Private Attributes

- **static GaussProcApproximation** *GPinstance*
  
  *pointer to the active object instance used within the static evaluator*

### 8.47.1 Detailed Description

Derived approximation class for Gaussian Process implementation.

The **GaussProcApproximation** class provides a global approximation (surrogate) based on a Gaussian process. The Gaussian process is built after normalizing the function values, with zero mean. Opt++ is used to determine the optimal values of the covariance parameters, those which minimize the negative log likelihood function.
8.47.2 Member Function Documentation

8.47.2.1 void GPmodel_apply (const RealVector & new_x, bool variance_flag, bool gradients_flag)

[private]

Function returns a response value using the GP surface.
The response value is computed at the design point specified by the RealVector function argument.
The documentation for this class was generated from the following files:

- GaussProcApproximation.H
- GaussProcApproximation.C
8.48 GenLaguerreOrthogPolynomial Class Reference

Derived orthogonal polynomial class for generalized Laguerre polynomials.

Inheritance diagram for GenLaguerreOrthogPolynomial::

```
<table>
<thead>
<tr>
<th>BasisPolynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrthogonalPolynomial</td>
</tr>
<tr>
<td>GenLaguerreOrthogPolynomial</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- `GenLaguerreOrthogPolynomial()`  
  default constructor

- `GenLaguerreOrthogPolynomial(const Real &alpha_stat)`  
  standard constructor

- `~GenLaguerreOrthogPolynomial()`  
  destructor

Protected Member Functions

- `const Real & get_value(const Real &x, unsigned short order)`  
  parameter x

- `const Real & get_gradient(const Real &x, unsigned short order)`  
  given parameter x

- `const Real & norm_squared(unsigned short order)`  
  return the inner product $\langle L^\alpha_n, L^\alpha_n \rangle = \|L^\alpha_n\|^2$

- `const RealArray & gauss_points(unsigned short order)`  
  corresponding to polynomial order n

- `const RealArray & gauss_weights(unsigned short order)`  
  corresponding to polynomial order n

- `const Real & weight_factor()`  
  calculate and return wtFactor based on alphaPoly
- void alpha_polynomial (const Real &alpha)
  set alphaPoly

- void alpha_stat (const Real &alpha)
  set alphaPoly using the conversion alphaPoly = alpha_stat - 1.

Private Attributes
- Real alphaPoly
  by Abramowitz and Stegun (differs from statistical PDF notation)

8.48.1 Detailed Description

Derived orthogonal polynomial class for generalized Laguerre polynomials.

The `GenLaguerreOrthogPolynomial` class evaluates a univariate generalized/associated Laguerre polynomial $L^\alpha_n$ of a particular order. These polynomials are orthogonal with respect to the weight function $x^\alpha \exp(-x)$ when integrated over the support range of $[0, +\infty]$. This corresponds to the probability density function $f(x) = x^\alpha \exp(-x) / \Gamma(alpha+1)$ for the standard gamma distribution, although common statistical PDF parameter conventions (see, e.g., the uncertain variables section in the DAKOTA Reference Manual) and the Abramowitz and Stegun orthogonal polynomial parameter conventions require an offset conversion in this case ($alpha\_poly = alpha\_stat - 1$ with the poly definition used in both cases above). It enables (mixed) multidimensional orthogonal polynomial basis functions within `OrthogPolyApproximation`. A special case is the `LaguerreOrthogPolynomial` (implemented separately), for which alpha\_poly = 0 and weight function = exp(-x) (the standard exponential distribution).

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

- GenLaguerreOrthogPolynomial.H
- GenLaguerreOrthogPolynomial.C
8.49 GetLongOpt Class Reference

(Advanced Computer Research Institute, Lyon, France).

Inheritance diagram for GetLongOpt::

```
GetLongOpt

CommandLineHandler
```

Public Types

- `enum OptType { Valueless, OptionalValue, MandatoryValue }`
  
  *enum for different types of values associated with command line options.*

Public Member Functions

- `GetLongOpt (const char optmark= '-')`
  
  *Constructor.*

- `~GetLongOpt ()`
  
  *Destructor.*

- `int parse (int argc, char *const argv)`
  
  *parse the command line args (argc, argv).*

- `int parse (char *const str, char *const p)`
  
  *parse a string of options (typically given from the environment).*

- `int enroll (const char *const opt, const OptType t, const char *const desc, const char *const val)`
  
  *Add an option to the list of valid command options.*

- `const char * retrieve (const char *const opt) const`
  
  *Retrieve value of option.*

- `void usage (std::ostream &outfile=Cout) const`
  
  *Print usage information to outfile.*

- `void usage (const char *str)`
  
  *Change header of usage output to str.*

- `void store (const char *name, const char *value)`
  
  *Store a specified option value.*
Private Member Functions

- char * basename (char *const p) const
  extract the base name from a string as delimited by '/'

- int setcell (Cell *c, char *valtoken, char *nexttoken, const char *p)
  internal convenience function for setting Cell::value

Private Attributes

- Cell * table
  option table

- const char * ustring
  usage message

- char * pname
  program basename

- char optmarker
  option marker

- int enroll_done
  finished enrolling

- Cell * last
  last entry in option table

8.49.1 Detailed Description

(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 "=".

8.49.2 Member Enumeration Documentation

8.49.2.1 enum OptType

enum for different types of values associated with command line options.

**Enumerator:**

Valueless option that may never have a value
**OptionalValue**  option with optional value

**MandatoryValue**  option with required value

### 8.49.3 Constructor & Destructor Documentation

#### 8.49.3.1 GetLongOpt (const char *optmark = ‘-‘)

Constructor.

Constructor for GetLongOpt takes an optional argument: the option marker. If unspecified, this defaults to ‘-‘, the standard (?) Unix option marker.

### 8.49.4 Member Function Documentation

#### 8.49.4.1 int parse (int argc, char *const *argv)

Parse the command line args (argc, argv).

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

#### 8.49.4.2 int parse (char *const str, char *const p)

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

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

#### 8.49.4.3 int enroll (const char *const opt, const OptType t, const char *const desc, const char *const val)

Add an option to the list of valid command options.

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

#### 8.49.4.4 const char * retrieve (const char *const opt) const

Retrieve value of option.

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

8.49.4.5 void usage (const char * str) [inline]

Change header of usage output to str.

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

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

- CommandLineHandler.H
- CommandLineHandler.C
8.50 Graphics Class Reference

for post-processing with Matlab, Tecplot, etc.

Public Member Functions

- `Graphics()`
  constructor

- `~Graphics()`
  destructor

- `void create_plots_2d(const Variables &vars, const Response &response)`
  creates the 2d graphics window and initializes the plots

- `void create_tabular_datastream(const Variables &vars, const Response &response, const String &tabular_data_file)`
  opens the tabular data file stream and prints the headings

- `void add_datapoint(const Variables &vars, const Response &response)`
  the tabular data file based on the results of a model evaluation

- `void add_datapoint(int i, double x, double y)`
  adds data to a single window in the 2d graphics

- `void new_dataset(int i)`
  for a single window in the 2d graphics

- `void show_data_3d(const RealVector &X, const RealVector &Y, const RealMatrix &F)`
  generate a new 3d plot for F(X,Y)

- `void close()`
  close graphics windows and tabular datastream

- `void set_x_labels2d(const char *x_label)`
  set x label for each plot equal to x_label

- `void set_y_labels2d(const char *y_label)`
  set y label for each plot equal to y_label

- `void set_x_label2d(int i, const char *x_label)`
  set x label for ith plot equal to x_label

- `void set_y_label2d(int i, const char *y_label)`
  set y label for ith plot equal to y_label
• void graphics_counter (int cntr)
  set graphicsCntr equal to cntr

• int graphics_counter () const
  return graphicsCntr

• void tabular_counter_label (const String &label)
  set tabularCntrLabel equal to label

Private Attributes

• Graphics2D * graphics2D
  pointer to the 2D graphics object

• bool win2dOn
  flag to indicate if 2D graphics window is active

• bool win3dOn
  flag to indicate if 3D graphics window is active

• bool tabularDataFlag
  flag to indicate if tabular data stream is active

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

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

• std::ofstream tabularDataFStream
  file stream for tabulation of graphics data within compute_response

8.50.1 Detailed Description

for post-processing with Matlab, Tecplot, etc.
There is only one Graphics object (dakotaGraphics) and it is global (for convenient access from strategies, models, and approximations).
8.50.2 Member Function Documentation

8.50.2.1 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).

8.50.2.2 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.

8.50.2.3 void add_datapoint (const Variables & vars, const Response & response)

the tabular data file based on the results of a model evaluation
Adds data to each 2d plot and each tabular data column (one for each active variable and for each response function). graphicsCntr is used for the x axis in the graphics and the first column in the tabular data.

8.50.2.4 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.

8.50.2.5 void new_dataset (int i)

for a single window in the 2d graphics
Used for displaying multiple data sets within the same plot.

8.50.2.6 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
8.51 GridApplicInterface Class Reference

using grid services such as Condor or Globus.

Inheritance diagram for GridApplicInterface::

```
  Interface
  ↓
ApplicationInterface
  ↓
GridApplicInterface
```

Public Member Functions

- **GridApplicInterface** (const ProblemDescDB &problem_db)
  
  *constructor*

- **~GridApplicInterface** ()
  
  *destructor*

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

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

- void **derived_synch** (PRPQueue &prp_queue)
  
  *classes. This version waits for at least one completion.*

- void **derived_synch_nowait** (PRPQueue &prp_queue)
  
  *any completions if none are immediately available.*

- int **derived_synchronous_local_analysis** (const int &analysis_id)
- const StringArray & **analysis_drivers** () const
  
  *retrieve the analysis drivers specification for application interfaces*

- const AnalysisCode * **analysis_code** () const
  
  *return AnalysisCode::fileNameMap when defined for derived Interface class*

Public Attributes

- **SysCallAnalysisCode code**
  
  *Used to read/write parameter files and responses.*
Protected Member Functions

- void derived_synch_kernel (PRPQueue &prp_queue)
  Convenience function for common code between wait and nowait case.

- bool grid_file_test (const String &root_file)
  test file(s) for existence based on root_file name

Protected Attributes

- IntSet idSet
  system call evaluations

- IntShortMap failCountMap
  map linking function evaluation id’s to number of response read failures

- start_grid_computing_t start_grid_computing
  handle to dynamically linked start_grid_computing function

- perform_analysis_t perform_analysis
  handle to dynamically linked perform_analysis grid function

- get_jobs_completed_t get_jobs_completed
  handle to dynamically linked get_jobs_completed grid function

- stop_grid_computing_t stop_grid_computing
  handle to dynamically linked stop_grid_computing function

8.51.1 Detailed Description

using grid services such as Condor or Globus.

This class is currently a modified copy of SysCallApplicInterface adapted for use with an external grid services library which was dynamically linked using dlopen() services.

8.51.2 Member Function Documentation

8.51.2.1 int derived_synchronous_local_analysis (const int & analysis_id) [inline, virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().

TODO - allow local analyses?????

Reimplemented from ApplicationInterface.

The documentation for this class was generated from the following files:
- GridApplicInterface.H
- GridApplicInterface.C
8.52 HermiteOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Hermite polynomials.

Inheritance diagram for HermiteOrthogPolynomial::

```
  BasisPolynomial
   |
   OrthogonalPolynomial
   |
HermiteOrthogPolynomial
```

**Public Member Functions**

- **HermiteOrthogPolynomial ()**
  
  default constructor

- **~HermiteOrthogPolynomial ()**
  
  destructor

**Protected Member Functions**

- **const Real & get_value (const Real &x, unsigned short order)**
  
  retrieve the Hermite polynomial value for a given parameter x

- **const Real & get_gradient (const Real &x, unsigned short order)**
  
  retrieve the Hermite polynomial gradient for a given parameter x

- **const Real & norm_squared (unsigned short order)**
  
  return the inner product $<He_n,He_n> = ||He_n||^2$

- **const RealArray & gauss_points (unsigned short order)**
  
  polynomial order

- **const RealArray & gauss_weights (unsigned short order)**
  
  polynomial order

**Static Private Attributes**

- **static const Real Pi**
  
  numerical value of Pi
8.52 HermiteOrthogPolynomial Class Reference

8.52.1 Detailed Description

Derived orthogonal polynomial class for Hermite polynomials.

The **HermiteOrthogPolynomial** class evaluates a univariate Hermite polynomial of a particular order. It uses the "probabilist's" formulation for which the polynomials are orthogonal with respect to the weight function \(1/\sqrt{2\pi} \exp(-x^2/2)\) when integrated over the support range of \([-\infty, +\infty]\). It enables (mixed) multidimensional orthogonal polynomial basis functions within **OrthogPolyApproximation**.

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

- HermiteOrthogPolynomial.H
- HermiteOrthogPolynomial.C
8.53 HierarchSurrModel Class Reference

Hierarchical surrogates (models of varying fidelity).

Inheritance diagram for HierarchSurrModel:

- Model
- SurrogateModel
- HierarchSurrModel

Public Member Functions

- HierarchSurrModel (ProblemDescDB &problem_db)
  constructor

- ~HierarchSurrModel ()
  destructor

Protected Member Functions

- void derived_compute_response (const ActiveSet &set)
  portion of compute_response() specific to HierarchSurrModel

- void derived_asynch_compute_response (const ActiveSet &set)
  portion of asynch_compute_response() specific to HierarchSurrModel

- const IntResponseMap & derived_synchronize ()
  portion of synchronize() specific to HierarchSurrModel

- const IntResponseMap & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to HierarchSurrModel

- Model & surrogate_model ()
  return lowFidelityModel

- Model & truth_model ()
  return highFidelityModel

- void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  return lowFidelityModel and highFidelityModel
- void **primary_response_fn_weights** (const RealVector &wts, bool recurse_flag=true)  
  squares terms and optionally recurses into LF/HF models

- void **surrogate_bypass** (bool bypass_flag)  
  for any lower-level surrogates.

- void **surrogate_function_indices** (const IntSet &surr_fn_indices)  
  (re)set the surrogate index set in SurrogateModel::surrogateFnIndices

- void **build_approximation** ()  
  correction of lowFidelityModel results

- void **component_parallel_mode** (short mode)  
  lowFidelityModel and highFidelityModel

- void **derived_init_communicators** (const int &max_iterator_concurrency, bool recurse_flag=true)  
  set up lowFidelityModel and highFidelityModel for parallel operations

- void **derived_init_serial** ()  
  set up lowFidelityModel and highFidelityModel for serial operations.

- void **derived_set_communicators** (const int &max_iterator_concurrency, bool recurse_flag=true)  
  highFidelityModel

- void **derived_free_communicators** (const int &max_iterator_concurrency, bool recurse_flag=true)  
  (request forwarded to lowFidelityModel and highFidelityModel)

- void **serve** ()  
  stop_servers().

- void **stop_servers** ()  
  **HierarchSurrModel** is complete.

- void **inactive_view** (short view, bool recurse_flag=true)  
  context and optionally recurse into

- int **evaluation_id** () const  
  Return the current evaluation id for the **HierarchSurrModel**.

- void **set_evaluation_reference** ()  
  (request forwarded to lowFidelityModel and highFidelityModel)

- void **fine_grained_evaluation_counters** ()  
  and highFidelityModel
void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const

(request forwarded to lowFidelityModel and highFidelityModel)

Private Member Functions

- void update_model (Model &model)
  
  with current variable values/bounds/labels

Private Attributes

- int hierModelEvals
  
  derived_asynch_compute_response()

- IntResponseMap cachedTruthRespMap
  
  portions were still pending.

- Model lowFidelityModel
  
  a data fit surrogate on a low fidelity model).

- Model highFidelityModel
  
  fidelity results. Model is of arbitrary type and supports recursions.

- Response highFidRefResponse
  
  and used for calculating corrections.

8.53.1 Detailed Description

Hierarchical surrogates (models of varying fidelity).

The HierarchSurrModel class manages hierarchical models of varying fidelity. In particular, it uses a low fidelity model as a surrogate for a high fidelity model. The class contains a lowFidelityModel which performs the approximate low fidelity function evaluations and a highFidelityModel which provides truth evaluations for computing corrections to the low fidelity results.

8.53.2 Member Function Documentation

8.53.2.1 void derived_compute_response (const ActiveSet & set) [protected, virtual]

portion of compute_response() specific to HierarchSurrModel
Compute the response synchronously using lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, compute the high fidelity response if needed with build_approximation(), and, if correction is active, correct the low fidelity results.

Reimplemented from Model.

8.53.2.2  void derived_asynch_compute_response (const ActiveSet & set)  [protected, virtual]

portion of async compute_response() specific to HierarchSurrModel

Compute the response asynchronously using lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, compute the high fidelity response with build_approximation() (for correcting the low fidelity results in derived_synchronize() and derived_synchronize_nowait()) if not performed previously.

Reimplemented from Model.

8.53.2.3  const IntResponseMap & derived_synchronize ()  [protected, virtual]

portion of synchronize() specific to HierarchSurrModel

Blocking retrieval of asynchronous evaluations from lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, apply correction (if active) to each response in the array. derived_synchronize() is designed for the general case where derived_asynch_compute_response() may be inconsistent in its use of low fidelity evaluations, high fidelity evaluations, or both.

Reimplemented from Model.

8.53.2.4  const IntResponseMap & derived_synchronize_nowait ()  [protected, virtual]

portion of synchronize_nowait() specific to HierarchSurrModel

Nonblocking retrieval of asynchronous evaluations from lowFidelityModel, highFidelityModel, or both (mixed case). For the lowFidelityModel portion, apply correction (if active) to each response in the map. derived_synchronize_nowait() is designed for the general case where derived_asynch_compute_response() may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from Model.

8.53.2.5  int evaluation_id () const  [inline, protected, virtual]

Return the current evaluation id for the HierarchSurrModel.

return the hierarchical model evaluation count. Due to possibly intermittent use of surrogate bypass, this is not the same as either the loFi or hiFi model evaluation counts. It also does not distinguish duplicate evals.

Reimplemented from Model.

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

- HierarchSurrModel.H
- HierarchSurrModel.C

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
8.54 HybridStrategy Class Reference

Base class for hybrid minimization strategies.

Inheritance diagram for HybridStrategy::

![Inheritance Diagram]

- **Protected Member Functions**
  - `HybridStrategy (ProblemDescDB &problem_db)`
    - *constructor*
  - `~HybridStrategy ()`
    - *destructor*
  - `void allocate_methods ()`
    - *initialize selectedIterators and userDefinedModels*
  - `void deallocate_methods ()`
    - *free communicators for selectedIterators and userDefinedModels*

- **Protected Attributes**
  - `StringArray methodList`
    - *the list of method identifiers*
  - `int numIterators`
    - *number of methods in methodList*
  - `IteratorArray selectedIterators`
    - *the set of iterators, one for each entry in methodList*
  - `ModelArray userDefinedModels`
    - *the set of models, one for each iterator*
8.54 HybridStrategy Class Reference

8.54.1 Detailed Description

Base class for hybrid minimization strategies.
This base class shares code for three approaches to hybrid minimization: (1) the sequential hybrid; (2) the embedded hybrid; and (3) the collaborative hybrid.

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

- HybridStrategy.H
- HybridStrategy.C
8.55 Interface Class Reference

Base class for the interface class hierarchy.

Inheritance diagram for Interface::

- Interface
  - ApplicationInterface
  - ApproximationInterface
  - DirectApplicInterface
  - ForkApplicInterface
  - GridApplicInterface
  - SysCallApplicInterface
  - ParallelDirectApplicInterface
  - SerialDirectApplicInterface

Public Member Functions

- **Interface ()**
  
  *default constructor*

- **Interface (ProblemDescDB & problem_db)**

  *standard constructor for envelope*

- **Interface (const Interface & interface)**

  *copy constructor*

- **virtual ~ Interface ()**

  *destructor*

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

  *assignment operator*

- **virtual void map (const Variables & vars, const ActiveSet & set, Response & response, const bool asynch_flag = false)**

  *variables to the responses.*

- **virtual const IntResponseMap & synch ()**

  *recovers data from a series of asynchronous evaluations (blocking)*

- **virtual const IntResponseMap & synch_nowait ()**

  *recovers data from a series of asynchronous evaluations (nonblocking)*

- **virtual void serve_evaluations ()**

  *evaluation server function for multiprocessor executions*

- **virtual void stop_evaluation_servers ()**

  *send messages from iterator rank 0 to terminate evaluation servers*
• virtual void `init_communicators` (const `IntArray` &message_lengths, const int &max_iterator_concurrency)

  iterator and concurrent multiprocessor analyses within an evaluation.

• virtual void `set_communicators` (const `IntArray` &message_lengths)

  (the partitions are already allocated in ParallelLibrary).

• virtual void `free_communicators` ()

  iterator and concurrent multiprocessor analyses within an evaluation.

• virtual void `init_serial` ()

  reset certain defaults for serial interface objects.

• virtual int `asynch_local_evaluation_concurrency` () const

  return the user-specified concurrency for asynch local evaluations

• virtual `String interface_synchronization` () const

  return the user-specified interface synchronization

• virtual int `minimum_samples` (bool constraint_flag) const

  `ApproximationInterface` (used by DataFitSurrModels).

• virtual int `recommended_samples` (bool constraint_flag) const

  `ApproximationInterface` (used by DataFitSurrModels).

• virtual void `approximation_function_indices` (const `IntSet` &approx_fn_indices)

  set the (currently active) approximation function index set

• virtual void `update_approximation` (const `Variables` &vars, const `Response` &response)

  updates the anchor point for an approximation

• virtual void `update_approximation` (const `VariablesArray` &vars_array, const `ResponseArray` &resp_array)

  updates the current data points for an approximation

• virtual void `append_approximation` (const `Variables` &vars, const `Response` &response)

  appends a single point to an existing approximation

• virtual void `append_approximation` (const `VariablesArray` &vars_array, const `ResponseArray` &resp_array)

  appends multiple points to an existing approximation

• virtual void `build_approximation` (const `BoolDeque` &rebuild_deque, const `RealVector` &lower_bnds, const `RealVector` &upper_bnds)

  builds the approximation
virtual void clear_current ()
  clears current data from an approximation interface

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

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

virtual const SurrogateDataPoint & anchor_point () const
  returns the anchorPoint used within an approximation interface

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

virtual const RealVectorArray & approximation_coefficients ()
  within an ApproximationInterface

virtual const StringArray & analysis_drivers () const
  retrieve the analysis drivers specification for application interfaces

virtual const AnalysisCode * analysis_code () const
  return AnalysisCode::fileNameMap when defined for derived Interface class

void assign_rep (Interface *interface_rep, bool ref_count_incr=true)
  replaces existing letter with a new one

const String & interface_type () const
  returns the interface type

const String & interface_id () const
  returns the interface identifier

int evaluation_id () const
returns the current function evaluation id for the interface

- void **fine_grained_evaluation_counters**(const size_t &num_fns) 
  set fineGrainEvalCounters to true and initialize counters if needed

- void **init_evaluation_counters**(const size_t &num_fns) 
  initialize fine grained evaluation counters

- void **set_evaluation_reference**() 
  set evaluation count reference points for the interface

- void **print_evaluation_summary**(std::ostream &s, bool minimal_header, bool relative_count) const 
  print an evaluation summary for the interface

- bool **multi_proc_eval_flag**() const 
  returns a flag signaling the use of multiprocessor evaluation partitions

- bool **iterator_eval_dedicated_master_flag**() const 
  iterator-evaluation scheduling level

- bool **is_null**() const 
  function to check interfaceRep (does this envelope contain a letter?)

### Protected Member Functions

- **Interface**(BaseConstructor, const ProblemDescDB &problem_db) 
  derived class constructors - Coplien, p. 139

- **Interface**(NoDBBaseConstructor, const size_t &num_fns) 
  (NoDBBaseConstructor used for on the fly instantiations without a DB)

- void **init_algebraic_mappings**(const Variables &vars, const Response &response) 
  Define algebraicACVIndices, algebraicACVIds, and algebraicFnIndices.

- void **asv_mapping**(const ActiveSet &total_set, ActiveSet &algebraic_set, ActiveSet &core_set) 
  from the total Interface evaluation requirements (total_set)

- void **algebraic_mappings**(const Variables &vars, const ActiveSet &algebraic_set, Response &algebraic_response) 
  and the data extracted from the algebraic_mappings file

- void **response_mapping**(const Response &algebraic_response, const Response &core_response, Response &total_response) 
  from derived_map() to create the total response
Protected Attributes

- **String interfaceType**
  the interface type: system, fork, direct, grid, or approximation

- **String idInterface**
  the interface specification identifier string from the DAKOTA input file

- **bool algebraicMappings**
  Interface’s parameter to response mapping that is explicit and algebraic.

- **bool coreMappings**
  ApplicationInterface or functionSurfaces for ApproximationInterface).

- **bool fineGrainEvalCounters**
  controls use of fn val/grad/hess counters

- **int fnEvalId**
  total interface evaluation counter

- **int newFnEvalId**
  new (non-duplicate) interface evaluation counter

- **int fnEvalIdRefPt**
  iteration reference point for fnEvalId

- **int newFnEvalIdRefPt**
  iteration reference point for newFnEvalId

- **IntArray fnValCounter**
  number of value evaluations by resp fn

- **IntArray fnGradCounter**
  number of gradient evaluations by resp fn

- **IntArray fnHessCounter**
  number of Hessian evaluations by resp fn

- **IntArray newFnValCounter**
  number of new value evaluations by resp fn

- **IntArray newFnGradCounter**
  number of new gradient evaluations by resp fn

- **IntArray newFnHessCounter**
  number of new Hessian evaluations by resp fn
- **IntArray fnValRefPt**
  iteration reference point for fnValCounter

- **IntArray fnGradRefPt**
  iteration reference point for fnGradCounter

- **IntArray fnHessRefPt**
  iteration reference point for fnHessCounter

- **IntArray newFnValRefPt**
  iteration reference point for newFnValCounter

- **IntArray newFnGradRefPt**
  iteration reference point for newFnGradCounter

- **IntArray newFnHessRefPt**
  iteration reference point for newFnHessCounter

- **IntResponseMap rawResponseMap**
  of asynchronous evaluations.

- **StringArray fnLabels**
  print_evaluation_summary() and derived direct interface classes)

- **bool multiProcEvalFlag**
  flag for multiprocessor evaluation partitions (evalComm)

- **bool ie DedMasterFlag**
  flag for dedicated master partitioning at the iterator level

- **short outputLevel**
  output verbosity level: [SILENT, QUIET, NORMAL, VERBOSE, DEBUG]_OUTPUT

### Private Member Functions

- **Interface * get_interface (ProblemDescDB &problem_db)**
  Used by the envelope to instantiate the correct letter class.

- **int algebraic_function_type (String)**
  evaluation call to make
Private Attributes

- **StringArray algebraicVarTags**  
  set of variable tags from AMPL stub.col

- **SizetArray algebraicACVIndices**  
  continuous variables

- **SizetArray algebraicACVIds**  
  continuous variables

- **StringArray algebraicFnTags**  
  set of function tags from AMPL stub.row

- **IntArray algebraicFnTypes**  
  AMPL objval (conival) calls.

- **SizetArray algebraicFnIndices**  
  DAKOTA response functions.

- **RealArray algebraicConstraintWeights**  
  set of weights for computing Hessian matrices for algebraic constraints;

- **int numAlgebraicResponses**  
  number of algebraic responses (objectives+constraints)

- **Interface * interfaceRep**  
  pointer to the letter (initialized only for the envelope)

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

- **ASL * asl**  
  pointer to an AMPL solver library (ASL) object

8.55.1 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.
8.55.2 Constructor & Destructor Documentation

8.55.2.1 Interface ()

default constructor
used in Model envelope class instantiations

8.55.2.2 Interface (ProblemDescDB & problem_db)

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

8.55.2.3 Interface (const Interface & interface)

copy constructor
Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.

8.55.2.4 ~Interface () [virtual]

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

8.55.2.5 Interface (BaseConstructor, const ProblemDescDB & problem_db) [protected]

derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all inherited interfaces. get_interface() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_interface() again). Since this is the letter and the letter IS the representation, interfaceRep is set to NULL (an uninitialized pointer causes problems in ~Interface).

8.55.3 Member Function Documentation

8.55.3.1 Interface operator= (const Interface & interface)

assignment operator
8.55.3.2 void assign_rep (Interface * interface_rep, bool ref_count_incr = true)

replaces existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old interfaceRep and assigns the new interfaceRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.

- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_interface(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

8.55.3.3 Interface * get_interface (ProblemDescDB & problem_db) [private]

Used by the envelope to instantiate the correct letter class.

used only by the envelope constructor to initialize interfaceRep to the appropriate derived type.

8.55.4 Member Data Documentation

8.55.4.1 IntResponseMap rawResponseMap [protected]

of asynchronous evaluations.

The map is a full/partial set of completions which are identified through their fnEvalId key. The raw set is postprocessed (i.e., finite difference gradients merged) in Model::synchronize() where it becomes responseMap.

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

- DakotaInterface.H
- DakotaInterface.C
**8.56 InterpPolyApproximation Class Reference**

approximation).

Inheritance diagram for InterpPolyApproximation::

```
Approximation

BasisPolyApproximation

InterpPolyApproximation
```

**Public Member Functions**

- **InterpPolyApproximation ()**
  
  default constructor

- **~InterpPolyApproximation ()**
  
  destructor

**Protected Member Functions**

- **int min_coefficients () const**
  
  build the derived class approximation type in numVars dimensions

- **void find_coefficients ()**
  
  interpolation polynomials

- **void allocate_arrays ()**
  
  size expansionCoeffs and expansionCoeffGrads

- **void compute_global_sensitivity ()**
  
  Performs global sensitivity analysis using Sobol’ Indices.

- **const Real & get_value (const RealVector &x)**
  
  retrieve the response expansion value for a given parameter vector

- **const RealVector & get_gradient (const RealVector &x)**
  
  and default DVV

- **const RealVector & get_gradient (const RealVector &x, const UIntArray &dvv)**
  
  and given DVV
• const Real & get_mean ()
  return the mean of the expansion, treating all variables as random

• const Real & get_mean (const RealVector &x)
  treating a subset of the variables as random

• RealVector get_mean_gradient ()
  treating all variables as random

• const RealVector & get_mean_gradient (const RealVector &x, const UIntArray &dvv)
  and given DVV, treating a subset of the variables as random

• const Real & get_variance ()
  return the variance of the expansion, treating all variables as random

• const Real & get_variance (const RealVector &x)
  treating a subset of the variables as random

• const RealVector & get_variance_gradient ()
  vector, treating all variables as random

• const RealVector & get_variance_gradient (const RealVector &x, const UIntArray &dvv)
  vector and given DVV, treating a subset of the variables as random

• const Real & get_covariance (const RealVector &exp_coeffs_2)
  return the covariance of the expansion, treating all variables as random

Private Member Functions

• const Real & tensor_product_value (const RealVector &x, size_t tp_index)
  tensor-product grid; contributes to get_value(x)

• const RealVector & tensor_product_gradient (const RealVector &x, size_t tp_index)
  tensor-product grid; contributes to get_gradient(x)

• const RealVector & tensor_product_gradient (const RealVector &x, size_t tp_index, const UIntArray &dvv)
  tensor-product grid for given DVV; contributes to get_gradient(x, dvv)

• const Real & tensor_product_mean (const RealVector &x, size_t tp_index)
  tensor-product grid; contributes to get_mean(x)

• const RealVector & tensor_product_mean_gradient (const RealVector &x, size_t tp_index, const UIntArray &dvv)
8.56 InterpPolyApproximation Class Reference

- tensor-product grid; contributes to get_mean(x)

- const Real & tensor_product_variance (const RealVector &x, size_t tp_index)
  tensor-product grid; contributes to get_variance(x)

- const RealVector & tensor_product_variance_gradient (const RealVector &x, size_t tp_index, const UIntArray &dvv)
  tensor-product grid; contributes to get_variance(x)

- void get_subsets ()
  performs sorting to store constituent subsets (constituentSets)

- void lower_sets (int plus_one_set, IntSet &top_level_set)
  recursively identifies constituent subsets

- Real partial_variance_integral (const int &set_value, size_t tp_index, UShortArray &quad_order)
  finds variance of interpolant with respect to variables in the set

- void partial_variance (const int &set_value)
  computes partialVariance

Private Attributes

- Array< IntSet > constituentSets
  the constituent subsets for each superset

- RealVector partialVariance
  the partialVariances of subset functions f_alpha

- Array< Array< BasisPolynomial > > polynomialBasis
  constructing the multivariate orthogonal/interpolation polynomials.

- int numCollocPts
  expansion (length of expansionCoeffs)

- UShort2DArray smolyakMultiIndex
  within the polynomialBasis for a particular variable

- RealArray smolyakCoeffs
  precomputed array of Smolyak combinatorial coefficients

- UShort3DArray collocKey
  the 1-D interpolant indices for sets of tensor-product collocation points.

- Size2DArray expansionCoeffIndices
set of tensor products to the expansionCoeffs array.

- Real tpValue
the value of a tensor-product interpolant; a contributor to approxValue

- RealVector tpGradient
  approxGradient

- Real tpMean
  the mean of a tensor-product interpolant; a contributor to expansionMean

- RealVector tpMeanGrad
  contributor to expansionMeanGrad

- Real tpVariance
  expansionVariance

- RealVector tpVarianceGrad
  contributor to expansionVarianceGrad

### 8.56.1 Detailed Description

approximation).

The InterpPolyApproximation class provides a global approximation based on interpolation polynomials. It is used primarily for stochastic collocation approaches to uncertainty quantification.

### 8.56.2 Member Function Documentation

#### 8.56.2.1 const Real & get_mean () [protected, virtual]

return the mean of the expansion, treating all variables as random

In this case, all expansion variables are random variables and the mean of the expansion is simply the sum over i of r_i w_i.

Implements BasisPolyApproximation.

#### 8.56.2.2 const Real & get_mean (const RealVector & x) [protected, virtual]

treating a subset of the variables as random

In this case, a subset of the expansion variables are random variables and the mean of the expansion involves evaluating the expectation over this subset.

Implements BasisPolyApproximation.
8.56.2.3 RealVector get_mean_gradient () [protected, virtual]

treating all variables as random

In this function, all expansion variables are random variables and any design/state variables are omitted from the expansion. In this case, the derivative of the expectation is the expectation of the derivative. The mixed derivative case (some design variables are inserted and some are augmented) requires no special treatment.

Implements BasisPolyApproximation.

8.56.2.4 const RealVector & get_mean_gradient (const RealVector & x, const UIntArray & dvv) [protected, virtual]

and given DVV, treating a subset of the variables as random

In this function, a subset of the expansion variables are random variables and any augmented design/state variables (i.e., not inserted as random variable distribution parameters) are included in the expansion. In this case, the mean of the expansion is the expectation over the random subset and the derivative of the mean is the derivative of the remaining expansion over the non-random subset. This function must handle the mixed case, where some design/state variables are augmented (and are part of the expansion: derivatives are evaluated as described above) and some are inserted (derivatives are obtained from expansionCoeffGrads).

Implements BasisPolyApproximation.

8.56.2.5 const Real & get_variance () [protected, virtual]

return the variance of the expansion, treating all variables as random

In this case, all expansion variables are random variables and the variance of the expansion is the sum over all but the first term of the coefficients squared times the polynomial norms squared.

Implements BasisPolyApproximation.

8.56.2.6 const Real & get_variance (const RealVector & x) [protected, virtual]

treating a subset of the variables as random

In this case, a subset of the expansion variables are random variables and the variance of the expansion involves summations over this subset.

Implements BasisPolyApproximation.

8.56.2.7 const RealVector & get_variance_gradient () [protected, virtual]

vector, treating all variables as random

In this function, all expansion variables are random variables and any design/state variables are omitted from the expansion. The mixed derivative case (some design variables are inserted and some are augmented) requires no special treatment.

Implements BasisPolyApproximation.
8.56.2.8  const RealVector & get_variance_gradient (const RealVector & x, const UIntArray & dvv)  
[protected, virtual]

vector and given DVV, treating a subset of the variables as random
In this function, a subset of the expansion variables are random variables and any augmented design/state variables
(i.e., not inserted as random variable distribution parameters) are included in the expansion. This function must handle
the mixed case, where some design/state variables are augmented (and are part of the expansion) and some are inserted
(derivatives are obtained from expansionCoeffGrads).
Implements BasisPolyApproximation.

8.56.2.9  void get_subsets ()  [private]

performs sorting to store constituent subsets (constituentSets)
Find constituent subsets.

8.56.2.10  void lower_sets (int plus_one_set, IntSet & top_level_set)  [private]

recursively identifies constituent subsets
For input set, recursively finds constituent subsets with one fewer element

8.56.2.11  Real partial_variance_integral (const int & set_value, size_t tp_index, UShortArray & quad_order)  [private]

finds variance of interpolant with respect to variables in the set
Forms an interpolant over variables that are members of the given set. Finds the variance of the interpolant w.r.t.
the variables in the set.

8.56.2.12  void partial_variance (const int & set_value)  [private]

computes partialVariance
Computes the partial expection for a certain set represented in integer form. Solves for lower level subsets if
necessary and stores all computations in subsetPartialVariance.

8.56.3  Member Data Documentation

8.56.3.1  Array< Array< BasisPolynomial >> polynomialBasis  [private]

constructing the multivariate orthogonal/interpolation polynomials.
Each variable (outer array size = numVars) may have multiple integration orders associated with it (inner array
size = num_levels_per_var = 1 for quadrature, w + numVars for sparse grid).
8.56 InterpPolyApproximation Class Reference 387

8.56.3.2 UShort2DArray smolyakMultiIndex [private]

within the polynomialBasis for a particular variable
The index sets correspond to \( j \) (0-based) for use as indices, which are offset from the \( i \) indices (1-based) normally used in the Smolyak expressions. For quadrature, the indices are zero (irrespective of integration order) since there is one polynomialBasis per variable; for sparse grid, the index corresponds to level - 1 within each anisotropic tensor-product integration of a Smolyak recursion.

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

- InterpPolyApproximation.H
- InterpPolyApproximation.C
8.57 Iterator Class Reference

Base class for the iterator class hierarchy.

Inheritance diagram for Iterator::

```
Iterator
  |__Analyzer
  |__Minimizer
  |__NonD
  |__PStudy
  |__DACE
  |__LeastSq
  |__Optimizer
  |__SurrBasedMinimizer
  |__NonDBayesCal
  |__NonDExpansion
  |__NonDIntegration
  |__NonDInterval
  |__NonDReliability
  |__NonDSampling
  |__DDACEDesignCompExp
  |__FSUDesignCompExp
  |__ParamStudy
  |__PSUADEDesignCompExp
  |__NL2SOLLeastSq
  |__NLSSOLLeastSq
  |__SNLLLeastSq
  |__APPSoptimizer
  |__COLINOptimizer
  |__CONMINOptimizer
  |__DOTOptimizer
  |__JEGAOptimizer
  |__NCSUOptimizer
  |__NLPQLPOptimizer
  |__NPSOLOptimizer
  |__SNLLOptimizer
  |__EffGlobalMinimizer
  |__SurrBasedGlobalMinimizer
  |__SurrBasedLocalMinimizer
```

Public Member Functions

- **Iterator ()**
  
  default constructor

- **Iterator (Model &model)**
  
  standard envelope constructor

- **Iterator (const String &method_name, Model &model)**
  
  alternate envelope constructor for instantiations by name

- **Iterator (const Iterator &iterator)**
  
  copy constructor

- **virtual ~Iterator ()**
  
  destructor

- **Iterator operator= (const Iterator &iterator)**
  
  assignment operator

- **virtual void pre_run ()**
  
  pre-run portion of run_iterator (optional)

- **virtual void run ()**
  
  and may contain pre/post steps in lieu of separate pre/post

- **virtual const Variables & variables_results () const**
  
  return a single final iterator solution (variables)

- **virtual const Response & response_results () const**
return a single final iterator solution (response)

- virtual bool accepts_multiple_points() const
  return is false. Override to return true if appropriate.

- virtual bool returns_multiple_points() const
  return is false. Override to return true if appropriate.

- virtual const VariablesArray & variables_array_results() const
  only be used if returns_multiple_points() returns true.

- virtual const ResponseArray & response_array_results() const
  only be used if returns_multiple_points() returns true.

- virtual void initial_points(const VariablesArray &pts)
  only be used if accepts_multiple_points() returns true.

- virtual void response_results_active_set(const ActiveSet &set)
  set the requested data for the final iterator response results.

- virtual void initialize_graphics(bool graph_2d, bool tabular_data, const String &tabular_file)
  initialize the 2D graphics window and the tabular graphics data

- virtual void print_results(std::ostream &s)
  print the final iterator results

- virtual void sampling_reset(int min_samples, int rec_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator

- virtual const String & sampling_scheme() const
  return sampling name

- virtual String uses_method() const
  return name of any enabling iterator used by this iterator

- virtual void method_recourse()
  perform a method switch, if possible, due to a detected conflict

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

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

- void initialize_run(std::ostream &s)
  utility function to verbosely perform common operations prior to run()
• void initialize_run ()
  utility function to quietly perform common operations prior to run()

• void run_iterator (std::ostream &s)
  verbosely

• void run_iterator ()
  utility function to automate initialize_run()/run()/finalize_run() quietly

• void post_run (std::ostream &s)
  post-run portion of run_iterator (optional); verbose to print results

• void post_run ()
  post-run portion of run_iterator (optional); quiet

• void finalize_run (std::ostream &s)
  utility function to verbosely perform common operations following run()

• void finalize_run ()
  utility function to quietly perform common operations following run()

• void assign_rep (Iterator *iterator_rep, bool ref_count_incr=true)
  replaces existing letter with a new one

• ProblemDescDB & problem_description_db () const
  return the problem description database (probDescDB)

• const String & method_name () const
  return the method name

• const String & method_id () const
  return the method identifier (idMethod)

• short output_level () const
  return the method output level (outputLevel)

• int maximum_concurrency () const
  return the maximum concurrency supported by the iterator

• void maximum_concurrency (int max_conc)
  set the maximum concurrency supported by the iterator

• void active_set (const ActiveSet &set)
  employ evaluate_parameter_sets())
• const ActiveSet & active_set () const
  employ evaluate_parameter_sets()

• void sub_iterator_flag (bool si_flag)
  set subIteratorFlag

• void variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &dr_target2)
  set primaryA{CV,DIV,DRV}MapIndices, secondaryA{CV,DIV,DRV}MapTargets

• bool is_null () const
  function to check iteratorRep (does this envelope contain a letter?)

• Iterator * iterator_rep () const
  that are not mapped to the top Iterator level

Protected Member Functions

• Iterator (BaseConstructor, Model &model)
  derived class constructors - Coplien, p. 139)

• Iterator (NoDBBaseConstructor, Model &model)
  alternate constructor for base iterator classes constructed on the fly

• Iterator (NoDBBaseConstructor)
  alternate constructor for base iterator classes constructed on the fly

• virtual void derived_initialize_run ()
  portions of initialize_run specific to derived iterators

• virtual void derived_post_run ()
  portions of post_run specific to derived iterators

• virtual void derived_finalize_run ()
  portions of finalize_run specific to derived iterators

• virtual const VariablesArray & initial_points () const
  be meaningful after a call to initial_points mutator.

Protected Attributes

• Model iteratedModel
  or a thin RecastModel wrapped around it
• **ProblemDescDB & probDescDB**
  class member reference to the problem description database

• **String methodName**
  name of the iterator (the user's method spec)

• **Real convergenceTol**
  iteration convergence tolerance

• **int maxIterations**
  maximum number of iterations for the iterator

• **int maxFunctionEvals**
  maximum number of fn evaluations for the iterator

• **int maxConcurrency**
  maximum coarse-grained concurrency

• **size_t numFunctions**
  number of response functions

• **size_t numContinuousVars**
  number of active continuous vars

• **size_t numDiscreteIntVars**
  number of active discrete integer vars

• **size_t numDiscreteRealVars**
  number of active discrete real vars

• **ActiveSet activeSet**
  tracks the response data requirements on each function evaluation

• **bool subIteratorFlag**
  *(NestedModel::subIterator or DataFitSurrModel::daceIterator)*

• **SizetArray primaryACVarMapIndices**
  from higher level iteration

• **SizetArray primaryADIVarMapIndices**
  higher level iteration

• **SizetArray primaryADRVarMapIndices**
  higher level iteration
- **ShortArray secondaryACVarMapTargets**  
  *from higher level iteration*

- **ShortArray secondaryADIVarMapTargets**  
  *from higher level iteration*

- **ShortArray secondaryADRVarMapTargets**  
  *from higher level iteration*

- **String gradientType**  
  *type of gradient data: analytic, numerical, mixed, or none*

- **String methodSource**  
  *source of numerical gradient routine: dakota or vendor*

- **String intervalType**  
  *type of numerical gradient interval: central or forward*

- **String hessianType**  
  *type of Hessian data: analytic, numerical, quasi, mixed, or none*

- **Real fdGradStepSize**  
  *relative finite difference step size for numerical gradients*

- **Real fdHessByGradStepSize**  
  *using first-order differences of gradients*

- **Real fdHessByFnStepSize**  
  *using second-order differences of function values*

- **short outputLevel**  
  *output verbosity level: {SILENT, QUIET, NORMAL, VERBOSE, DEBUG}_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.*

- **Iterator * get_iterator (const String &method_name, Model &model)**  
  *Used by the envelope to instantiate the correct letter class.*

- **void pre_output ()**
convenience function to write variables to file, following pre-run

- virtual void post_input ()
  read tabular data for post-run mode

Private Attributes

- String idMethod
  method identifier string from the input file

- Iterator * iteratorRep
  pointer to the letter (initialized only for the envelope)

- int referenceCount
  number of objects sharing iteratorRep

8.57.1 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.

8.57.2 Constructor & Destructor Documentation

8.57.2.1 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.

8.57.2.2 Iterator (Model & model)

standard envelope constructor

Used in iterator instantiations within strategy constructors. Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data.
8.57 Iterator Class Reference

8.57.2.3  **Iterator (const String & method_name, Model & model)**

alternate envelope constructor for instantiations by name

Used in sub-iterator instantiations within iterator constructors. Envelope constructor only needs to extract enough data to properly execute `get_iterator()`, since letter holds the actual base class data.

8.57.2.4  **Iterator (const Iterator & iterator)**

copy constructor

Copy constructor manages sharing of iteratorRep and incrementing of referenceCount.

8.57.2.5  **~Iterator ()** [virtual]

destructor

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

8.57.2.6  **Iterator (BaseConstructor, Model & model)** [protected]

derived class constructors - Coplien, p. 139)

This constructor builds the base class data for all inherited iterators. `get_iterator()` instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling `get_iterator()` again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in `~Iterator`).

8.57.2.7  **Iterator (NoDBBaseConstructor, Model & model)** [protected]

alternate constructor for base iterator classes constructed on the fly

This alternate constructor builds base class data for inherited iterators. It is used for on-the-fly instantiations for which DB queries cannot be used. Therefore it only sets attributes taken from the incoming model.

8.57.2.8  **Iterator (NoDBBaseConstructor)** [protected]

alternate constructor for base iterator classes constructed on the fly

This alternate constructor builds base class data for inherited iterators. It is used for on-the-fly instantiations for which DB queries cannot be used. It has no incoming model, so only sets up a minimal set of defaults. However, its use is preferable to the default constructor, which should remain as minimal as possible.

8.57.3  **Member Function Documentation**

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

assignment operator

8.57.3.2 void pre_run () [virtual]

pre-run portion of run_iterator (optional)

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely contained in the derived run function

Reimplemented in DDACEDesignCompExp, FSUDesignCompExp, NonDLHSSampling, ParamStudy, and PSUADEDesignCompExp.

8.57.3.3 void run () [virtual]

and may contain pre/post steps in lieu of separate pre/post

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented in LeastSq, NonD, Optimizer, PStudyDACE, and SurrBasedMinimizer.

8.57.3.4 void initialize_graphics (bool graph_2d, bool tabular_data, const String & tabular_file) [virtual]

initialize the 2D graphics window and the tabular graphics data

This is a convenience function for encapsulating graphics initialization operations. It does not require a strategy-Rep forward since it is only used by letter objects.

Reimplemented in NonDReliability, and SurrBasedMinimizer.

8.57.3.5 void print_results (std::ostream & s) [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented in Analyzer, LeastSq, Optimizer, PStudyDACE, NonDBayesCal, NonDExpansion, NonDGlobalReliability, NonDIncrementLHSSampling, NonDInterval, NonDLHSSampling, NonDLocalReliability, NonDPolynomialChaos, and SurrBasedMinimizer.

8.57.3.6 void initialize_run (std::ostream & s)

utility function to verbosely perform common operations prior to run()

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is one form of the overloaded initialize-run function. This form accepts an ostream and executes verbosely. It is used for standard stand-alone iterator executions. This function is not virtual: derived portions are defined in derived_initialize_-run().
8.57.3.7  void initialize_run ()

utility function to quietly perform common operations prior to run()

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is one form of the overloaded initialize-run function. This form does not accept an ostream and executes quietly. It is commonly used in sub-iterator executions. This function is not virtual: derived portions are defined in derived_initialize_run().

8.57.3.8  void run_iterator (std::ostream & s)

verbosely

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This non-virtual function is one form of the overloaded run_iterator function which automates the initialize-run/run/finalize-run portions of the progression. This form accepts an ostream and executes verbosely.

8.57.3.9  void run_iterator ()

utility function to automate initialize_run()/run()/finalize_run() quietly

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This non-virtual function is one form of the overloaded run_iterator function which automates the initialize-run/run/finalize-run portions of the progression. This form does not accept an ostream and executes quietly.

8.57.3.10  void post_run (std::ostream & s)

post-run portion of run_iterator (optional); verbose to print results

post-run phase, which a derived iterator may optionally reimplement; when not present, post-run activities are likely in run

8.57.3.11  void post_run ()

post-run portion of run_iterator (optional); quiet

post-run phase, which a derived iterator may optionally implement; when not present, post-run activities are likely in run

8.57.3.12  void finalize_run (std::ostream & s)

utility function to verbosely perform common operations following run()

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is one form of the overloaded finalize-run function. This form accepts an ostream and executes verbosely. This function is not virtual: derived portions are defined in derived_finalize_run().
8.57.3.13 void finalize_run ()

utility function to quietly perform common operations following run()

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is one form of the overloaded finalize-run function. This form does not accept an ostream and executes quietly. This function is not virtual: derived portions are defined in derived_finalize_run().

8.57.3.14 void assign_rep (Iterator ∗ iterator_rep, bool ref_count_incr = true)

replaces existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old iteratorRep and assigns the new iteratorRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.

- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_iterator(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

8.57.3.15 void derived_initialize_run () [protected, virtual]

portions of initialize_run specific to derived iterators

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of initialize_run(). Redefinition by derived classes is optional.

Reimplemented in CONMINOptimizer, LeastSq, Minimizer, NonD, Optimizer, DOTOptimizer, NLPQLPOptimizer, SNLLLeastSq, and SNLLOptimizer.

8.57.3.16 void derived_post_run () [protected, virtual]

portions of post_run specific to derived iterators

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of post_run(). Redefinition by derived classes is optional.

Reimplemented in LeastSq, Optimizer, DDACEDesignCompExp, FSUDesignCompExp, NonDLHSSampling, ParamStudy, PSUADesignCompExp, SNLLLeastSq, and SNLLOptimizer.

8.57.3.17 void derived_finalize_run () [protected, virtual]

portions of finalize_run specific to derived iterators
8.57 Iterator Class Reference

**Iterator** supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of `finalize_run()`. Redefinition by derived classes is optional.

Reimplemented in `LeastSq`, `Minimizer`, `NonD`, `Optimizer`, `SNLLLeastSq`, and `SNLLOptimizer`.

8.57.3.18 **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.

8.57.3.19 **Iterator** ∗ get_iterator (const String & method_name, Model & model) [private]

Used by the envelope to instantiate the correct letter class.

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

8.57.4 Member Data Documentation

8.57.4.1 **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.

8.57.4.2 **Real** fdHessByGradStepSize [protected]

using first-order differences of gradients

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

8.57.4.3 **Real** fdHessByFnStepSize [protected]

using second-order differences of function values

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

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

- DakotaIterator.H
- DakotaIterator.C
8.58 JacobiOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Jacobi polynomials.

Inheritance diagram for JacobiOrthogPolynomial::

```
BasisPolynomial
    OrthogonalPolynomial
        JacobiOrthogPolynomial
```

Public Member Functions

- **JacobiOrthogPolynomial ()**
  default constructor

- **JacobiOrthogPolynomial (const Real &alpha_stat, const Real &beta_stat)**
  standard constructor

- **~JacobiOrthogPolynomial ()**
  destructor

Protected Member Functions

- const Real & **get_value** (const Real &x, unsigned short order)
  retrieve the Jacobi polynomial value for a given parameter x

- const Real & **get_gradient** (const Real &x, unsigned short order)
  retrieve the Jacobi polynomial gradient for a given parameter x

- const Real & **norm_squared** (unsigned short order)
  \[ ||P^\alpha(\alpha,\beta)_n||^2 \]

- const RealArray & **gauss_points** (unsigned short order)
  polynomial order n

- const RealArray & **gauss_weights** (unsigned short order)
  polynomial order n

- const Real & **weight_factor ()**
  calculate and return wtFactor based on alphaPoly and betaPoly
8.58 JacobiOrthogPolynomial Class Reference

- void alpha_polynomial (const Real &alpha)
  
  set alphaPoly

- void beta_polynomial (const Real &beta)
  
  set betaPoly

- void alpha_stat (const Real &alpha)
  
  set betaPoly using the conversion betaPoly = alpha_stat - 1.

- void beta_stat (const Real &beta)
  
  set alphaPoly using the conversion alphaPoly = beta_stat - 1.

Private Attributes

- Real alphaPoly
  
  Abramowitz and Stegun (differs from statistical PDF notation).

- Real betaPoly
  
  Abramowitz and Stegun (differs from statistical PDF notation).

8.58.1 Detailed Description

Derived orthogonal polynomial class for Jacobi polynomials.

The JacobiOrthogPolynomial class evaluates a univariate Jacobi polynomial \( P^\alpha(\alpha,\beta)_n \) of a particular order. These polynomials are orthogonal with respect to the weight function \((1-x)^\alpha(1+x)^\beta\) when integrated over the support range of \([-1,+1]\). This corresponds to the probability density function \( f(x) = (1-x)^\alpha(1+x)^\beta / (2^{\alpha+\beta+1} B(\alpha+1,\beta+1)) \) for the beta distribution for \([L,U]=[-1,1]\), where common statistical PDF notation conventions (see, e.g., the uncertain variables section in the DAKOTA Reference Manual) and the Abramowitz and Stegun orthogonal polynomial conventions are inverted and require conversion in this case (alpha_poly = beta_stat - 1; beta_poly = alpha_stat - 1 with the poly definitions used in both cases above). It enables (mixed) multidimensional orthogonal polynomial basis functions within OrthogPolyApproximation. A special case is the LegendreOrthogPolynomial (implemented separately), for which alpha_poly = beta_poly = 0.

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

- JacobiOrthogPolynomial.H
- JacobiOrthogPolynomial.C
8.59 JEGAOptimizer Class Reference

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

Inheritance diagram for JEGAOptimizer::

```
Iterator
    
Minimizer
    
Optimizer
    
JEGAOptimizer
```

Public Member Functions

- virtual void find_optimum ()
  
  "Performs the iterations to determine the optimal set of solutions."

- virtual bool accepts_multiple_points () const
  
  "Overridden to return true since JEGA algorithms can accept multiple initial points."

- virtual bool returns_multiple_points () const
  
  "Overridden to return true since JEGA algorithms can return multiple final points."

- virtual void initial_points (const VariablesArray &pts)
  
  "Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables."

- virtual const VariablesArray & initial_points () const
  
  "Overridden to return the collection of initial points for the JEGA algorithm created and run by this JEGAOptimizer."

- JEGAOptimizer (Model &model)
  
  "Constructs a JEGAOptimizer class object."

- ~JEGAOptimizer ()
  
  "Destructs a JEGAOptimizer."

Protected Member Functions

- void LoadDakotaResponses (const JEGA::Utilities::Design &from, Variables &vars, Response &resp) const
  
  "Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects."
- void ReCreateTheParameterDatabase ()
  
  Destroys the current parameter database and creates a new empty one.

- void LoadTheParameterDatabase ()
  
  Reads information out of the known Dakota::ProblemDescDB and puts it into the current parameter database.

- void LoadAlgorithmConfig (JEGA::FrontEnd::AlgorithmConfig &aConfig)
  
  Completely initializes the supplied algorithm configuration.

- void LoadProblemConfig (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Completely initializes the supplied problem configuration.

- void LoadTheDesignVariables (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Adds DesignVariableInfo objects into the problem configuration object.

- void LoadTheObjectiveFunctions (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Adds ObjectiveFunctionInfo objects into the problem configuration object.

- void LoadTheConstraints (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Adds ConstraintInfo objects into the problem configuration object.

- const JEGA::Utilities::Design * GetBestSolution (const JEGA::Utilities::DesignOFSortSet &from)
  
  Chooses the best Design from a set of solutions taking into account the algorithm type.

- const JEGA::Utilities::Design * GetBestMOSolution (const JEGA::Utilities::DesignOFSortSet &from)
  
  Chooses the best Design from a set of solutions assuming that they are generated by a multi objective algorithm.

- const JEGA::Utilities::Design * GetBestSOSolution (const JEGA::Utilities::DesignOFSortSet &from)
  
  Chooses the best Design from a set of solutions assuming that they are generated by a single objective algorithm.

- JEGA::DoubleMatrix ToDoubleMatrix (const VariablesArray &variables) const
  
  Converts the items in a VariablesArray into a DoubleMatrix whereby the items in the matrix are the design variables.

- void resize_variables_results_array (std::size_t newsize)
  
  Safely resizes the best variables array taking into account the requirements put forth by the envelope-letter design pattern.

- void resize_response_results_array (std::size_t newsize)
  
  Safely resizes the best response array taking into account the requirements put forth by the envelope-letter design pattern.
Private Attributes

- EvaluatorCreator * _theEvalCreator
  A pointer to an EvaluatorCreator used to create the evaluator used by JEGA in Dakota (a JEGAEvaluator).

- JEGA::Utilities::ParameterDatabase * _theParamDB
  A pointer to the ParameterDatabase from which all parameters are retrieved by the created algorithms.

- VariablesArray _initPts
  An array of initial points to use as an initial population.

Static Private Attributes

- static const std::string SOGA_METHOD_TXT
  The text that indicates the SOGA method.

- static const std::string MOGA_METHOD_TXT
  The text that indicates the MOGA method.

Classes

- class Driver
  A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.

- class Evaluator
  An evaluator specialization that knows how to interact with Dakota.

- class EvaluatorCreator
  A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

8.59.1 Detailed Description

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

This class encapsulates the necessary functionality for creating and properly initializing the JEGA algorithms (MOGA and SOGA).

8.59.2 Constructor & Destructor Documentation

8.59.2.1 JEGAOptimizer (Model & model)

Constructs a JEGAOptimizer class object.

This method does some of the initialization work for the algorithm. In particular, it initialized the JEGA core.
Parameters:

\[model\] The Dakota::Model that will be used by this optimizer for problem information, etc.

### 8.59.3 Member Function Documentation

#### 8.59.3.1 void LoadDakotaResponses (const JEGA::Utilities::Design & from, Variables & vars, Response & resp) const [protected]

Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects. This version is meant for the case where a Variables and a Response object exist and just need to be loaded.

Parameters:

- \[from\] The JEGA Design class object from which to extract the variable and response information for Dakota.
- \[vars\] The Dakota::Variables object into which to load the design variable values of \[from\].
- \[resp\] The Dakota::Response object into which to load the objective function and constraint values of \[from\].

#### 8.59.3.2 void LoadTheParameterDatabase () [protected]

Reads information out of the known Dakota::ProblemDescDB and puts it into the current parameter database. This should be called from the JEGAOptimizer constructor since it is the only time when the problem description database is certain to be configured to supply data for this optimizer.

#### 8.59.3.3 void LoadAlgorithmConfig (JEGA::FrontEnd::AlgorithmConfig & aConfig) [protected]

Completely initializes the supplied algorithm configuration. This loads the supplied configuration object with appropriate data retrieved from the parameter database.

Parameters:

- \[aConfig\] The algorithm configuration object to load.

#### 8.59.3.4 void LoadProblemConfig (JEGA::FrontEnd::ProblemConfig & pConfig) [protected]

Completely initializes the supplied problem configuration. This loads the fresh configuration object using the LoadTheDesignVariables, LoadTheObjectiveFunctions, and LoadTheConstraints methods.

Parameters:

- \[pConfig\] The problem configuration object to load.
void LoadTheDesignVariables (JEGA::FrontEnd::ProblemConfig & pConfig) [protected]

Adds DesignVariableInfo objects into the problem configuration object.
This retrieves design variable information from the ParameterDatabase and creates DesignVariableInfo’s from it.

Parameters:

\[ pConfig \] The problem configuration object to load.

void LoadTheObjectiveFunctions (JEGA::FrontEnd::ProblemConfig & pConfig) [protected]

Adds ObjectiveFunctionInfo objects into the problem configuration object.
This retrieves objective function information from the ParameterDatabase and creates ObjectiveFunctionInfo’s from it.

Parameters:

\[ pConfig \] The problem configuration object to load.

void LoadTheConstraints (JEGA::FrontEnd::ProblemConfig & pConfig) [protected]

Adds ConstraintInfo objects into the problem configuration object.
This retrieves constraint function information from the ParameterDatabase and creates ConstraintInfo’s from it.

Parameters:

\[ pConfig \] The problem configuration object to load.

const Design * GetBestSolution (const JEGA::Utilities::DesignOFSortSet & from) [protected]

Chooses the best Design from a set of solutions taking into account the algorithm type.
eventually this functionality must be moved into a separate post-processing application for MO datasets.

const Design * GetBestMOSolution (const JEGA::Utilities::DesignOFSortSet & from) [protected]

Chooses the best Design from a set of solutions assuming that they are generated by a multi objective algorithm.
eventually this functionality must be moved into a separate post-processing application for MO datasets.
8.59.3.10 const Design * GetBestSOSolution (const JEGA::Utilities::DesignOFSortSet & from) [protected]

Chooses the best Design from a set of solutions assuming that they are generated by a single objective algorithm. Eventually this functionality must be moved into a separate post-processing application for MO datasets.

8.59.3.11 JEGA::DoubleMatrix ToDoubleMatrix (const VariablesArray & variables) const [protected]

Converts the items in a VariablesArray into a DoubleMatrix whereby the items in the matrix are the design variables. The matrix will not contain responses but when being used by Dakota, this doesn’t matter. JEGA will attempt to re-evaluate these points but Dakota will recognize that they do not require re-evaluation and thus it will be a cheap operation.

Parameters:

variables The array of DakotaVariables objects to use as the contents of the returned matrix.

Returns:

The matrix created using the supplied VariablesArray.

8.59.3.12 void resize_variables_results_array (std::size_t newsize) [protected]

Safely resizes the best variables array taking into account the requirements put forth by the envelope-letter design pattern. Do not directly call resize on the bestVariablesArray object unless you intend to share the internal content (letter) with other objects after assignment.

Parameters:

newsize The new size for the variables array.

8.59.3.13 void resize_response_results_array (std::size_t newsize) [protected]

Safely resizes the best response array taking into account the requirements put forth by the envelope-letter design pattern. Do not directly call resize on the bestResponseArray object unless you intend to share the internal content (letter) with other objects after assignment.

Parameters:

newsize The new size for the responses array.
8.59.3.14 **void find_optimum ()** [virtual]

Performs the iterations to determine the optimal set of solutions.

Override of pure virtual method in **Optimizer** base class.

The extraction of parameter values actually occurs in this method when the JEGA::FrontEnd::Driver::Execute-Algorithm is called. Also the loading of the problem and algorithm configurations occurs in this method. That way, if it is called more than once and the algorithm or problem has changed, it will be accounted for.

Implements **Optimizer**.

8.59.3.15 **bool accepts_multiple_points () const** [virtual]

Overridden to return true since JEGA algorithms can accept multiple initial points.

**Returns:**
true, always.

Reimplemented from **Iterator**.

8.59.3.16 **bool returns_multiple_points () const** [virtual]

Overridden to return true since JEGA algorithms can return multiple final points.

**Returns:**
true, always.

Reimplemented from **Iterator**.

8.59.3.17 **void initial_points (const **VariablesArray** & pts) [virtual]

Overridden to assign the _initPts member variable to the passed in collection of **Dakota::Variables**.

**Parameters:**
- **pts** The array of initial points for the JEGA algorithm created and run by this **JEGAOptimizer**.

Reimplemented from **Iterator**.

8.59.3.18 **const **VariablesArray** & initial_points () const** [virtual]

Overridden to return the collection of initial points for the JEGA algorithm created and run by this **JEGAOptimizer**.

**Returns:**
- The collection of initial points for the JEGA algorithm created and run by this **JEGAOptimizer**.

Reimplemented from **Iterator**.
8.59.4 Member Data Documentation

8.59.4.1 VariablesArray _initPts [private]

An array of initial points to use as an initial population.

This member is here to help support the use of JEGA algorithms in Dakota strategies. If this array is populated, then whatever initializer is specified will be ignored and the DoubleMatrix initializer will be used instead on a matrix created from the data in this array.

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

- JEGAOptimizer.H
- JEGAOptimizer.C
8.60 JEGAOptimizer::Driver Class Reference

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

Public Member Functions

- GeneticAlgorithm * ExtractAllData (const AlgorithmConfig &algConfig)
  Reads all required data from the problem description database stored in the supplied algorithm config.

- DesignOFSortSet PerformIterations (GeneticAlgorithm *theGA)
  Performs the required iterations on the supplied GA.

- void DestroyAlgorithm (GeneticAlgorithm *theGA)
  Deletes the supplied GA.

- Driver (const ProblemConfig &probConfig)
  Default constructs a Driver.

8.60.1 Detailed Description

A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm. This is necessary because DAKOTA requires that all problem information be extracted from the problem description DB at the time of Optimizer construction and the front end does it all in the execute algorithm method which must be called in find_optimum.

8.60.2 Constructor & Destructor Documentation

8.60.2.1 Driver (const ProblemConfig & probConfig) [inline]

Default constructs a Driver.

Parameters:

- probConfig  The definition of the problem to be solved by this Driver whenever ExecuteAlgorithm is called.

The problem can be solved in multiple ways by multiple algorithms even using multiple different evaluators by issuing multiple calls to ExecuteAlgorithm with different AlgorithmConfigs.

8.60.3 Member Function Documentation

8.60.3.1 GeneticAlgorithm* ExtractAllData (const AlgorithmConfig & algConfig) [inline]

Reads all required data from the problem description database stored in the supplied algorithm config.
The returned GA is fully configured and ready to be run. It must also be destroyed at some later time. You MUST call DestroyAlgorithm for this purpose. Failure to do so could result in a memory leak and an eventual segmentation fault! Be sure to call DestroyAlgorithm prior to destroying the algorithm config that was used to create it!

This is just here to expose the base class method to users.

Parameters:

- **algConfig** The fully loaded configuration object containing the database of parameters for the algorithm to be run on the known problem.

Returns:

The fully configured and loaded GA ready to be run using the PerformIterations method.

### 8.60.3.2 DesignOFSortSet PerformIterations (GeneticAlgorithm * theGA) [inline]

Performs the required iterations on the supplied GA.
This includes the calls to AlgorithmInitialize and AlgorithmFinalize and logs some information if appropriate.
This is just here to expose the base class method to users.

Parameters:

- **theGA** The GA on which to perform iterations. This parameter must be non-null.

Returns:

The final solutions reported by the supplied GA after all iterations and call to AlgorithmFinalize.

### 8.60.3.3 void DestroyAlgorithm (GeneticAlgorithm * theGA) [inline]

Deletes the supplied GA.
Use this method to destroy a GA after all iterations have been run. This method knows if the log associated with the GA was created here and needs to be destroyed as well or not.
This is just here to expose the base class method to users.
Be sure to use this prior to destroying the algorithm config object which contains the target. The GA destructor needs the target to be in tact.

Parameters:

- **theGA** The algorithm that is no longer needed and thus must be destroyed.

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

- JEGAOptimizer.C
An evaluator specialization that knows how to interact with Dakota.

Public Member Functions

- virtual bool Evaluate (DesignGroup &group)
  
  *Does evaluation of each design in group.*

- virtual bool Evaluate (Design &des)
  
  *This method cannot be used!!*

- virtual std::string GetName () const
  
  *Returns the proper name of this operator.*

- virtual std::string GetDescription () const
  
  *Returns a full description of what this operator does and how.*

- virtual GeneticAlgorithmOperator * Clone (GeneticAlgorithm &algorithm) const
  
  *Creates and returns a pointer to an exact duplicate of this operator.*

- Evaluator (GeneticAlgorithm &algorithm, Model &model)
  
  *Constructs a Evaluator for use by algorithm.*

- Evaluator (const Evaluator &copy)
  
  *Copy constructs a Evaluator.*

- Evaluator (const Evaluator &copy, GeneticAlgorithm &algorithm, Model &model)
  
  *Copy constructs a Evaluator for use by algorithm.*

Static Public Member Functions

- static const std::string & Name ()
  
  *Returns the proper name of this operator.*

- static const std::string & Description ()
  
  *Returns a full description of what this operator does and how.*

Protected Member Functions

- void SeparateVariables (const Design &from, RealVector &intoCont, IntVector &intoDiscInt, RealVector &intoDiscReal) const
  
  *This method fills intoCont, intoDiscInt and intoDiscReal appropriately using the values of from.*
8.61 JEGAOptimizer::Evaluator Class Reference

- void RecordResponses (const RealVector &from, Design &into) const
  
  Records the computed objective and constraint function values into into.

- std::size_t GetNumberNonLinearConstraints () const
  
  Returns the number of non-linear constraints for the problem.

- std::size_t GetNumberLinearConstraints () const
  
  Returns the number of linear constraints for the problem.

Private Member Functions

- Evaluator (GeneticAlgorithm &algorithm)
  
  This constructor has no implementation and cannot be used.

Private Attributes

- Model & _model
  
  The Model known by this evaluator.

8.61.1 Detailed Description

An evaluator specialization that knows how to interact with Dakota.
This evaluator knows how to use the model to do evaluations both in synchronous and asynchronous modes.

8.61.2 Constructor & Destructor Documentation

8.61.2.1 Evaluator (GeneticAlgorithm & algorithm, Model & model) [inline]

Constructs a Evaluator for use by algorithm.
The optimizer is needed for purposes of variable scaling.

Parameters:

  algorithm The GA for which the new evaluator is to be used.

  model The model through which evaluations will be done.
8.61.2.2 **Evaluator** (const **Evaluator** & *copy*)  [inline]

Copy constructs a **Evaluator**.

**Parameters:**

*copy* The evaluator from which properties are to be duplicated into this.

8.61.2.3 **Evaluator** (const **Evaluator** & *copy*, **GeneticAlgorithm** & *algorithm*, **Model** & *model*)  [inline]

Copy constructs a **Evaluator** for use by *algorithm*.

The optimizer is needed for purposes of variable scaling.

**Parameters:**

*copy* The existing **Evaluator** from which to retrieve properties.

*algorithm* The GA for which the new evaluator is to be used.

*model* The model through which evaluations will be done.

8.61.2.4 **Evaluator** (**GeneticAlgorithm** & *algorithm*)  [private]

This constructor has no implementation and cannot be used.

This constructor can never be used. It is provided so that this operator can still be registered in an operator registry even though it can never be instantiated from there.

**Parameters:**

*algorithm* The GA for which the new evaluator is to be used.

8.61.3 **Member Function Documentation**

8.61.3.1 static const std::string& **Name** ()  [inline, static]

Returns the proper name of this operator.

**Returns:**

The string "DAKOTA JEGA Evaluator".

8.61.3.2 static const std::string& **Description** ()  [inline, static]

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

The returned text is:
This evaluator uses Sandia’s DAKOTA optimization software to evaluate the passed in Designs. This makes it possible to take advantage of the fact that DAKOTA is designed to run on massively parallel machines.

Returns:
A description of the operation of this operator.

8.61.3.3 void SeparateVariables (const Design & from, RealVector & intoCont, IntVector & intoDiscInt, RealVector & intoDiscReal) const [protected]

This method fills intoCont, intoDiscInt and intoDiscReal appropriately using the values of from.

The discrete integer design variable values are placed in intoDiscInt, the discrete real design variable values are placed in intoDiscReal, and the continuum are placed into intoCont. The values are written into the vectors from the beginning so any previous contents of the vectors will be overwritten.

Parameters:
from The Design class object from which to extract the discrete design variable values.
intoDiscInt The vector into which to place the extracted discrete integer values.
intoDiscReal The vector into which to place the extracted discrete real values.
intoCont The vector into which to place the extracted continuous values.

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

Records the computed objective and constraint function values into into.

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

The response vector from is expected to contain values for each objective function followed by values for each non-linear constraint in the order in which the info objects were loaded into the target by the optimizer class.

Parameters:
from The vector of responses to install into into.
into The Design to which the responses belong and into which they must be written.

8.61.3.5 std::size_t GetNumberNonLinearConstraints () const [inline, protected]

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

This is computed by adding the number of non-linear equality constraints to the number of non-linear inequality constraints. These values are obtained from the model.
Returns:

The total number of non-linear constraints.

8.61.3.6 std::size_t GetNumberLinearConstraints () const [inline, protected]

Returns the number of linear constraints for the problem.
This is computed by adding the number of linear equality constraints to the number of linear inequality constraints.
These values are obtained from the model.

Returns:

The total number of linear constraints.

8.61.3.7 bool Evaluate (DesignGroup & group) [virtual]

Does evaluation of each design in group.
This method uses the Model known by this class to get Designs evaluated. It properly formats the Design class information in a way that Dakota will understand and then interprets the Dakota results and puts them back into the Design class object. It respects the asynchronous flag in the Model so evaluations may occur synchronously or asynchronously.
Prior to evaluating a Design, this class checks to see if it is marked as already evaluated. If it is, then the evaluation of that Design is not carried out. This is not strictly necessary because Dakota keeps track of evaluated designs and does not re-evaluate. An exception is the case of a population read in from a file complete with responses where Dakota is unaware of the evaluations.

Parameters:

 group  The group of Design class objects to be evaluated.

Returns:

true if all evaluations completed and false otherwise.

8.61.3.8 virtual bool Evaluate (Design & des) [inline, virtual]

This method cannot be used!!
This method does nothing and cannot be called. This is because in the case of asynchronous evaluation, this method would be unable to conform. It would require that each evaluation be done in a synchronous fashion.

Parameters:

 des  A Design that would be evaluated if this method worked.

Returns:

Would return true if the Design were evaluated and false otherwise. Never actually returns here. Issues a fatal error. Otherwise, it would always return false.
8.61.3.9  virtual std::string GetName () const  [inline, virtual]

Returns the proper name of this operator.

Returns:
See Name().

8.61.3.10 virtual std::string GetDescription () const  [inline, virtual]

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

Returns:
See Description().

8.61.3.11 virtual GeneticAlgorithmOperator* Clone (GeneticAlgorithm & algorithm) const  [inline, virtual]

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

Parameters:
algorithm The GA for which the clone is being created.

Returns:
A clone of this operator.

8.61.4 Member Data Documentation

8.61.4.1 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 file:

- JEGAOptimizer.C
8.62 JEGAOptimizer::EvaluatorCreator Class Reference

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

Public Member Functions

- virtual GeneticAlgorithmEvaluator * CreateEvaluator (GeneticAlgorithm &alg)
  
  **Overriden to return a newly created Evaluator.**

- EvaluatorCreator (Model &theModel)
  
  **Constructs an EvaluatorCreator using the supplied model.**

Private Attributes

- Model & _theModel
  
  **The user defined model to be passed to the constructor of the Evaluator.**

8.62.1 Detailed Description

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

8.62.2 Constructor & Destructor Documentation

8.62.2.1 EvaluatorCreator (Model & theModel) [inline]

Constructs an EvaluatorCreator using the supplied model.

Parameters:

- theModel  The Dakota::Model this creator will pass to the created evaluator.

8.62.3 Member Function Documentation

8.62.3.1 virtual GeneticAlgorithmEvaluator* CreateEvaluator (GeneticAlgorithm & alg) [inline, virtual]

Overriden to return a newly created Evaluator.

The GA will assume ownership of the evaluator so we needn’t worry about keeping track of it for destruction.

The additional parameters needed by the Evaluator are stored as members of this class at construction time.

Parameters:

- alg  The GA for which the evaluator is to be created.
Returns:

A pointer to a newly created Evaluator.

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

- JEGAOptimizer.C
8.63 LagrangeInterpPolynomial Class Reference

Derived basis polynomial class for 1-D Lagrange interpolation polynomials.

Inheritance diagram for LagrangeInterpPolynomial:

```
BasisPolynomial
  
LagrangeInterpPolynomial
```

Public Member Functions

- `LagrangeInterpPolynomial()`
  default constructor

- `LagrangeInterpPolynomial(const RealArray &interpolation_pts)`
  standard constructor

- `~LagrangeInterpPolynomial()`
  destructor

- `const Real & get_value(const Real &x, unsigned short i)`
  parameter x

- `const Real & get_gradient(const Real &x, unsigned short i)`
  given parameter x

- `void interpolation_points(const RealArray &interpolation_pts)`
  set interpolationPts

Private Member Functions

- `void precompute_data()`
  precompute data that is reused repeatedly within Lagrange interpolation

Private Attributes

- `RealArray interpolationPts`
  evaluated at the j_th interpolation point produces Kronecker delta_ij

- `size_t numInterpPts`
number of 1-D interpolation points

- RealVector lagDenominators
  precompute_data()

8.63.1 Detailed Description

Derived basis polynomial class for 1-D Lagrange interpolation polynomials.

The `LagrangeInterpPolynomial` class evaluates a univariate Lagrange interpolation polynomial. The order of the polynomial is dictated by the number of interpolation points \( \text{order} = N_p - 1 \). It enables multidimensional interpolants within `InterpPolyApproximation`.

8.63.2 Member Function Documentation

8.63.2.1 const Real & get_value (const Real & x, unsigned short i) [virtual]

parameter x
Compute value of Lagrange polynomial for interpolation point i.
Reimplemented from `BasisPolynomial`.

8.63.2.2 const Real & get_gradient (const Real & x, unsigned short i) [virtual]

given parameter x
Compute derivative with respect to x of Lagrange polynomial for interpolation point i.
Reimplemented from `BasisPolynomial`.

8.63.2.3 void precompute_data () [private]

precompute data that is reused repeatedly within Lagrange interpolation
Pre-compute denominator products that are only a function of the interpolationPts.
The documentation for this class was generated from the following files:

- LagrangeInterpPolynomial.H
- LagrangeInterpPolynomial.C
8.64 LaguerreOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Laguerre polynomials.

Inheritance diagram for LaguerreOrthogPolynomial::

```
BasisPolynomial
   |
   v
OrthogonalPolynomial
   |
   v
LaguerreOrthogPolynomial
```

### Public Member Functions

- **LaguerreOrthogPolynomial ()**
  
  *default constructor*

- **~LaguerreOrthogPolynomial ()**
  
  *destructor*

### Protected Member Functions

- **const Real & get_value (const Real &x, unsigned short order)**
  
  *retrieve the Laguerre polynomial value for a given parameter x*

- **const Real & get_gradient (const Real &x, unsigned short order)**
  
  *retrieve the Laguerre polynomial gradient for a given parameter x*

- **const Real & norm_squared (unsigned short order)**
  
  *return the inner product $<L_n, L_n> = ||L_n||^2$*

- **const RealArray & gauss_points (unsigned short order)**
  
  *polynomial order n*

- **const RealArray & gauss_weights (unsigned short order)**
  
  *polynomial order n*

### 8.64.1 Detailed Description

Derived orthogonal polynomial class for Laguerre polynomials.
The `LaguerreOrthogPolynomial` class evaluates a univariate Laguerre polynomial of a particular order. These polynomials are orthogonal with respect to the weight function \( \exp(-x) \) when integrated over the support range of \([0, +\infty)\). This corresponds to the probability density function for the standard exponential distribution. It enables (mixed) multidimensional orthogonal polynomial basis functions within `OrthogPolyApproximation`. Laguerre polynomials are a special case (\( \alpha = 0 \)) of the generalized Laguerre polynomials (implemented separately) which correspond to the standard gamma distribution.

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

- `LaguerreOrthogPolynomial.H`
- `LaguerreOrthogPolynomial.C`
8.65 LeastSq Class Reference

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

Inheritance diagram for LeastSq:

```
Iterator
    |   |
    |   |
    Minimizer
    |   |
    |   |
    LeastSq
```

Protected Member Functions

- **LeastSq ()**
  *default constructor*

- **LeastSq (Model &model)**
  *standard constructor*

- **LeastSq (NoDBBaseConstructor, Model &model)**
  *alternate constructor*

- **~LeastSq ()**
  *destructor*

- void derived_initialize_run ()
- void run ()
  *and may contain pre/post steps in lieu of separate pre/post*

- void derived_post_run ()
- void derived_finalize_run ()
  *portions of finalize_run specific to derived iterators*

- void print_results (std::ostream &s)
- virtual void minimize_residuals ()=0
  *for the least squares branch.*

- void read_observed_data ()
  *read user data file to load observed data points*
• void get_confidence_intervals ()
  Calculate confidence intervals on estimated parameters.

Static Protected Member Functions

• static void primary_resp_recast (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &scaled_response)
  (user) to iterator space

Protected Attributes

• int numLeastSqTerms
  number of least squares terms

• LeastSq * prevLSqInstance
  pointer containing previous value of leastSqInstance

• bool weightFlag
  flag indicating whether weighted least squares is active

• String obsDataFilename
  filename from which to read observed data

• bool obsDataFlag
  flag indicating whether user-supplied data is active

• RealArray obsData
  storage for user-supplied data for computing residuals

• RealVector confBoundsLower
  lower bounds for confidence intervals on calibration parameters

• RealVector confBoundsUpper
  upper bounds for confidence intervals on calibration parameters

Static Protected Attributes

• static LeastSq * leastSqInstance
  pointer to LeastSq instance used in static member functions
8.65.1 Detailed Description

Base class for the nonlinear least squares branch of the iterator hierarchy.
The LeastSq class provides common data and functionality for least squares solvers (including NL2OL, NLSSOLLeastSq, and SNLLLLeastSq).

8.65.2 Constructor & Destructor Documentation

8.65.2.1 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.

8.65.3 Member Function Documentation

8.65.3.1 void derived_initialize_run () [protected, virtual]

This function should be invoked (or reimplemented) by any derived implementations of derived_initialize_run() (which would otherwise hide it).
Reimplemented from Minimizer.
Reimplemented in SNLLLLeastSq.

8.65.3.2 void run () [inline, protected, virtual]

and may contain pre/post steps in lieu of separate pre/post
Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.

8.65.3.3 void derived_post_run () [protected, virtual]

Implements portions of post_run specific to LeastSq for scaling back to native variables and functions. This function should be invoked (or reimplemented) by any derived implementations of derived_post_run() (which would otherwise hide it).
Reimplemented from Iterator.
Reimplemented in SNLLLLeastSq.

8.65.3.4 void derived_finalize_run () [inline, protected, virtual]

portions of finalize_run specific to derived iterators
Iterator supports a construct/initialize-run/run finalize-run/destruct progression. This function is the virtual derived class portion of finalize_run(). Redefinition by derived classes is optional.

Redefinition by derived classes is optional.

Reimplemented in SNLLLeastSq.

8.65.3.5 \textbf{void print_results (std::ostream & s) [protected, virtual]}

Redefines default iterator results printing to include nonlinear least squares results (residual terms and constraints).

Reimplemented from Iterator.

8.65.3.6 \textbf{void primaryRespRecast (const Variables & native_vars, const Variables & scaled_var, const Response & native_response, Response \& iterator_response) [static, protected]}

(user) to iterator space

Least squares function map from user/native space to iterator/scaled space using a RecastModel. If no scaling also copies constraints.

8.65.3.7 \textbf{void readObservedData () [protected]}

read user data file to load observed data points

read user's observation data for computation of least squares residuals (currently reading on all processors – need to read once and broadcast)

8.65.3.8 \textbf{void getConfidenceIntervals () [protected]}

Calculate confidence intervals on estimated parameters.

Calculate individual confidence intervals for each parameter. These bounds are based on a linear approximation of the nonlinear model.

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

- DakotaLeastSq.H
- DakotaLeastSq.C

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
8.66 LegendreOrthogPolynomial Class Reference

Derived orthogonal polynomial class for Legendre polynomials.

Inheritance diagram for LegendreOrthogPolynomial:

Public Member Functions

- LegendreOrthogPolynomial() /default constructor
- ~LegendreOrthogPolynomial() /destructor

Protected Member Functions

- const Real & get_value(const Real &, unsigned short order) /retrieve the Legendre polynomial value for a given parameter x
- const Real & get_gradient(const Real &, unsigned short order) /retrieve the Legendre polynomial gradient for a given parameter x
- const Real & norm_squared(unsigned short order) /return the inner product <P_n,P_n> = ||P_n||^2
- const RealArray & gauss_points(unsigned short order) /polynomial order n
- const RealArray & gauss_weights(unsigned short order) /polynomial order n

8.66.1 Detailed Description

Derived orthogonal polynomial class for Legendre polynomials.
The `LegendreOrthogPolynomial` class evaluates a univariate Legendre polynomial of a particular order. These polynomials are orthogonal with respect to the weight function 1 when integrated over the support range of \([-1,+1]\). This corresponds to the probability density function \(f(x) = 1/(U-L) = 1/2\) for the uniform distribution for \([L,U]=[-1,1]\). It enables (mixed) multidimensional orthogonal polynomial basis functions within OrthogPolyApproximation. Legendre polynomials are a special case (\(alpha = beta = 0\)) of the more general Jacobi polynomials (implemented separately) which correspond to the beta distribution.

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

- `LegendreOrthogPolynomial.H`
- `LegendreOrthogPolynomial.C`
8.67 List Class Template Reference

Template class for the Dakota bookkeeping list.

Public Member Functions

- **List ()**
  
  Default constructor.

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

- **~List ()**
  
  Destructor.

- **template<class InputIter> List (InputIter first, InputIter last)**
  
  Range constructor (member template).

- **List< T > & operator= (const List< T > &a)**
  
  Assignment operator

- **void write (std::ostream &s) const**
  
  Writes a List to an output stream.

- **void read (MPIUnpackBuffer &s)**
  
  Reads a List from an MPIUnpackBuffer after an MPI receive.

- **void write (MPIPackBuffer &s) const**
  
  Writes a List to a MPIPackBuffer prior to an MPI send.

- **size_t entries () const**
  
  Returns the number of items that are currently in the list.

- **T get ()**
  
  Removes and returns the first item in the list.

- **T removeAt (size_t index)**
  
  Removes and returns the item at the specified index.

- **bool remove (const T &a)**
  
  Removes the specified item from the list.

- **void insert (const T &a)**
  
  Adds the item a to the end of the list.
8.67 List Class Template Reference

- **bool contains** (const T &a) const
  
  Returns TRUE if list contains object a, returns FALSE otherwise.

- **bool find** (bool(*test_fn)(const T &, const void *), const void *test_fn_data, T &found_item) const
  
  Finds and sets k to this object

- **List<T>::iterator find** (bool(*test_fn)(const T &, const void *), const void *test_fn_data)
  
  Returns an iterator pointing to an object that the test function finds.

- **size_t index** (bool(*test_fn)(const T &, const void *), const void *test_fn_data) const
  
  Returns the index of object that the test function finds.

- **size_t index** (const T &a) const
  
  Returns the index of the object.

- **size_t count** (bool(*test_fn)(const T &, const void *), const void *test_fn_data) const
  
  Returns the number of items in the list that satisfy the test function.

8.67.1 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 and extends the STL list class to add Dakota specific methods. Builds upon the previously existing DakotaValList class

8.67.2 Member Function Documentation

8.67.2.1 **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.

8.67.2.2 **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.
8.67.2.3  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.

8.67.2.4  void insert (const T & a)  [inline]

Adds the item a to the end of the list.
Insert item at end of list, calls list::push_back() method.

8.67.2.5  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.

8.67.2.6  bool find (bool(*)(const T &, const void * test_fn, const void * test_fn_data, T & found_item) const

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.

8.67.2.7  List<T>::iterator find (bool(*)(const T &, const void * test_fn, const void * test_fn_data)

Returns an iterator pointing to an object that the test function finds.
Find the first item in the list which satisfies the test function and return an iterator pointing to it.

8.67.2.8  size_t index (bool(*)(const T &, const void * test_fn, const void * test_fn_data) const

Returns the index of object that the 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).

8.67.2.9  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).

The documentation for this class was generated from the following file:
• DakotaList.H
### 8.68 MergedConstraints Class Reference

de the merged data view.

Inheritance diagram for MergedConstraints::

```
    Constraints
     |       |
     |       |
MergedConstraints
```

#### Public Member Functions

- **MergedConstraints ()**
  
  *default constructor*

- **MergedConstraints (const ProblemDescDB &problem_db, const std::pair< short, short > &view, const Sizet2DArray &vars_comps)**
  
  *standard constructor*

- **~MergedConstraints ()**
  
  *destructor*

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

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

#### Protected Member Functions

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

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

- **void build_active_views ()**
  
  *construct active views of all variables bounds arrays*

- **void build_inactive_views ()**
  
  *construct inactive views of all variables bounds arrays*
8.68 MergedConstraints Class Reference

8.68.1 Detailed Description

the merged data view.

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MergedConstraints derived class combines continuous and discrete domain types but separates design, uncertain, and state variable types. The result is merged design bounds arrays (mergedDesignLowerBnds, mergedDesignUpperBnds), uncertain distribution bounds arrays (uncertain-LowerBnds, uncertainUpperBnds), and merged state bounds arrays (mergedStateLowerBnds, mergedStateUpperBnds). The branch and bound strategy uses this approach (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to the Constraints constructor in Model).

8.68.2 Constructor & Destructor Documentation

8.68.2.1 MergedConstraints (const ProblemDescDB & problem_db, const std::pair< short, short > & view, const Sizet2DArray & vars_comps)

standard constructor

In this class, a merged data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators/strategies which use this class include: BranchBndOptimizer. Extract fundamental lower and upper bounds and merge continuous and discrete domains to create mergedDesignLowerBnds, merged-DesignUpperBnds, mergedStateLowerBnds, and mergedStateUpperBnds.

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

- MergedConstraints.H
- MergedConstraints.C
8.69 MergedVariables Class Reference

merged data view.

Inheritance diagram for MergedVariables::

```
  Variables
  |
  MergedVariables
```

Public Member Functions

- **MergedVariables ()**
  *default constructor*

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

- **~MergedVariables ()**
  *destructor*

- **const UIntArray &merged_discrete_ids () const**
  *returns the list of discrete variables merged into a continuous array*

- **void read (std::istream &s)**
  *read a variables object from an std::istream*

- **void write (std::ostream &s) const**
  *write a variables object to an std::ostream*

- **void write_aprepro (std::ostream &s) const**
  *write a variables object to an std::ostream in aprepro format*

- **void read_tabular (std::istream &s)**
- **void write_tabular (std::ostream &s) const**
  *write a variables object in tabular format to an std::ostream*

Protected Member Functions

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

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

- **void build_active_views** ()
  
  construct active views of all variables arrays

- **void build_inactive_views** ()
  
  construct inactive views of all variables arrays

**Private Attributes**

- **UIntArray mergedDiscreteIds**
  
  requirement is relaxed by merging them into a continuous array

### 8.69.1 Detailed Description

merged data view.

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MergedVariables derived class combines continuous and discrete domain types but separates design, uncertain, and state variable types. The result is a single continuous array of design variables (mergedDesignVars), a single continuous array of uncertain variables (uncertainVars), and a single continuous array of state variables (mergedStateVars). The branch and bound strategy uses this approach (see Variables::get_-variables(problem_db)).

### 8.69.2 Constructor & Destructor Documentation

#### 8.69.2.1 MergedVariables (const ProblemDescDB & problem_db, const std::pair< short, short > & view)

standard constructor

In this class, a merged data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators/strategies which use this class include: BranchBndOptimizer. Extract fundamental variable types and labels and merge continuous and discrete domains to create aggregate arrays and views.

### 8.69.3 Member Function Documentation

#### 8.69.3.1 void read_tabular (std::istream & s) [virtual]

Presumes variables object is appropriately sized to receive data

Reimplemented from Variables.

The documentation for this class was generated from the following files:
- MergedVariables.H
- MergedVariables.C
8.70 Minimizer Class Reference

iterator hierarchy.

Inheritance diagram for Minimizer::

Public Member Functions

- const Variables & variables_results () const
  return a single final iterator solution (variables)

- const Response & response_results () const
  return a single final iterator solution (response)

- const VariablesArray & variables_array_results () const
  only be used if returns_multiple_points() returns true.

- const ResponseArray & response_array_results () const
  only be used if returns_multiple_points() returns true.

- void response_results_active_set (const ActiveSet &set)
  set the requested data for the final iterator response results

Protected Member Functions

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

• Minimizer (NoDBBaseConstructor, Model &model)
  alternate constructor for "on the fly" instantiations

• Minimizer (NoDBBaseConstructor, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq)
  alternate constructor for "on the fly" instantiations

• ~Minimizer ()
  destructor

• void derived_initialize_run ()
  portions of initialize_run specific to derived iterators

• void derived_finalize_run ()
  portions of finalize_run specific to derived iterators

• void initialize_scaling ()
  checking

• void compute_scaling (int object_type, int auto_type, int num_vars, RealVector &lbs, RealVector &ubs, RealVector &targets, const StringArray &scale_strings, const RealVector &scales, IntArray &scale_types, RealVector &scale_mults, RealVector &scale_offsets)
  vector of variables, functions, constraints, etc.

• bool compute_scale_factor (const Real lower_bound, const Real upper_bound, Real *multiplier, Real *offset)
  automatically compute a single scaling factor – bounds case

• bool compute_scale_factor (const Real target, Real *multiplier)
  automatically compute a single scaling factor – target case

• bool need_resp_trans_byvars (const ShortArray &asv, int start_index, int num_resp)
  transformations

• RealVector modify_n2s (const RealVector &native_vars, const IntArray &scale_types, const RealVector &multipliers, const RealVector &offsets) const
  general RealVector mapping from native to scaled variables vectors:

• RealVector modify_s2n (const RealVector &scaled_vars, const IntArray &scale_types, const RealVector &multipliers, const RealVector &offsets) const
  general RealVector mapping from scaled to native variables (and values)

• void response_modify_n2s (const Variables &scaled_vars, const Response &native_response, Response &scaled_response, int native_offset, int recast_offset, int num_responses) const
map responses from native to scaled variable space

- void \texttt{response\_modify\_s2n} (const \texttt{Variables} &native\_vars, const \texttt{Response} &scaled\_response, \texttt{Response} &native\_response, int scaled\_offset, int native\_offset, int num\_responses) const

map responses from scaled to native variable space

- \texttt{RealMatrix lin\_coeffs\_modify\_n2s} (const \texttt{RealMatrix} &native\_coeffs, const \texttt{RealVector} &cv\_multipliers, const \texttt{RealVector} &lin\_multipliers) const

general linear coefficients mapping from native to scaled space

- void \texttt{print\_scaling} (const \texttt{String} &info, const \texttt{IntArray} &scale\_types, const \texttt{RealVector} &scale\_mults, const \texttt{RealVector} &scale\_offsets, const \texttt{StringArray} &labels)

print scaling information for a particular response type in tabular form

\section*{Static Protected Member Functions}

- static void \texttt{variables\_recast} (const \texttt{Variables} &scaled\_vars, \texttt{Variables} &native\_vars)

variables from scaled to native (user) space

- static void \texttt{secondary\_resp\_recast} (const \texttt{Variables} &native\_vars, const \texttt{Variables} &scaled\_vars, const \texttt{Response} &native\_response, \texttt{Response} &scaled\_response)

transform constraints (fns, grads, Hessians) from native (user) to

\section*{Protected Attributes}

- Real \texttt{constraintTol}

optimizer/least squares constraint tolerance

- Real \texttt{bigRealBoundSize}

cutoff value for inequality constraint and continuous variable bounds

- int \texttt{bigIntBoundSize}

cutoff value for discrete variable bounds

- size\_t \texttt{numNonlinearIneqConstraints}

number of nonlinear inequality constraints

- size\_t \texttt{numNonlinearEqConstraints}

number of nonlinear equality constraints

- size\_t \texttt{numLinearIneqConstraints}

number of linear inequality constraints

- size\_t \texttt{numLinearEqConstraints}
- int numNonlinearConstraints
  total number of nonlinear constraints

- int numLinearConstraints
  total number of linear constraints

- int numConstraints
  total number of linear and nonlinear constraints

- bool boundConstraintFlag
  constraints. Used for method selection and error checking.

- bool speculativeFlag
  flag for speculative gradient evaluations

- size_t numUserPrimaryFns
  number of objective functions or least squares terms in the user’s model

- size_t numIterPrimaryFns
  number of objective functions or least squares terms in iterator’s view

- bool scaleFlag
  flag for overall scaling status

- bool varsScaleFlag
  flag for variables scaling

- bool primaryRespScaleFlag
  flag for primary response scaling

- bool secondaryRespScaleFlag
  flag for secondary response scaling

- IntArray cvScaleTypes
  scale flags for continuous vars.

- RealVector cvScaleMultipliers
  scales for continuous variables

- RealVector cvScaleOffsets
  offsets for continuous variables

- IntArray responseScaleTypes
  scale flags for all responses
- RealVector `responseScaleMultipliers`
  scales for all responses

- RealVector `responseScaleOffsets`
  offsets for all responses (zero for functions, not for nonlin con)

- IntArray `linearIneqScaleTypes`
  scale flags for linear ineq

- RealVector `linearIneqScaleMultipliers`
  scales for linear ineq constrs.

- RealVector `linearIneqScaleOffsets`
  offsets for linear ineq constrs.

- IntArray `linearEqScaleTypes`
  scale flags for linear eq.

- RealVector `linearEqScaleMultipliers`
  scales for linear constraints

- RealVector `linearEqScaleOffsets`
  offsets for linear constraints

- Minimizer * `prevMinInstance`
  pointer containing previous value of minimizerInstance

- bool `vendorNumericalGradFlag`
  convenience flag for gradType == numerical && methodSource == vendor

- Variables `bestVariables`
  best variables found in minimization

- Response `bestResponse`
  best response found in minimization

- VariablesArray `bestVariablesArray`
  collection of all best solution variables.

- ResponseArray `bestResponseArray`
  collection of all best solution responses.
Static Protected Attributes

- static Minimizer * minimizerInstance
  
  pointer to Minimizer used in static member functions

Friends

- class SOLBase
  
  access to iterator hierarchy data (to avoid attribute replication)

- class SNLLBase
  
  access to iterator hierarchy data (to avoid attribute replication)

8.70.1 Detailed Description

iterator hierarchy.
The Minimizer class provides common data and functionality for Optimizer and LeastSq.

8.70.2 Constructor & Destructor Documentation

8.70.2.1 Minimizer (Model & model) [protected]

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

8.70.3 Member Function Documentation

8.70.3.1 void derived_initialize_run () [protected, virtual]

portions of initialize_run specific to derived iterators
Iterator supports a construct/initialize-run/run/finalize-run/destroy progression. This function is the virtual derived class portion of initialize_run(). Redefinition by derived classes is optional.
Reimplemented from Iterator.
Reimplemented in CONMINOptimizer, LeastSq, Optimizer, DOTOptimizer, NLPQLPOptimizer, SNLLLeastSq, and SNLLOptimizer.

8.70.3.2 void derived_finalize_run () [inline, protected, virtual]

portions of finalize_run specific to derived iterators
Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of finalize_run(). Redefinition by derived classes is optional.

Reimplemented from Iterator.

Reimplemented in LeastSq, Optimizer, SNLLLeastSq, and SNLLOptimizer.

8.70.3.3 void initialize_scaling () [protected]

checking
helper function used in constructors of derived classes to set up scaling types, multipliers and offsets when input scaling flag is enabled

8.70.3.4 void variables_recast (const Variables & scaled_vars, Variables & native_vars) [static, protected]

variables from scaled to native (user) space
Variables map from iterator/scaled space to user/native space using a RecastModel.

8.70.3.5 void secondary_resp_recast (const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response) [static, protected]

transform constraints (fns, grads, Hessians) from native (user) to
Constraint function map from user/native space to iterator/scaled/combined space using a RecastModel.

8.70.3.6 bool need_resp_trans_byvars (const ShortArray & asv, int start_index, int num_resp) [protected]

transformations
Determine if variable transformations present and derivatives requested, which implies a response transformation is necessay

8.70.3.7 RealVector modify_n2s (const RealVector & native_vars, const IntArray & scale_types, const RealVector & multipliers, const RealVector & offsets) const [protected]

general RealVector mapping from native to scaled variables vectors:
general RealVector mapping from native to scaled variables; loosely, in greatest generality: scaled_var = log( (native_var - offset) / multiplier )

8.70.3.8 RealVector modify_s2n (const RealVector & scaled_vars, const IntArray & scale_types, const RealVector & multipliers, const RealVector & offsets) const [protected]

general RealVector mapping from scaled to native variables (and values)
general real vector mapping from scaled to native variables and/or vals; loosely, in greatest generality: scaled_var = \((\text{LOG\_BASE}^\text{3}\text{scaled\_var}) \ast \text{multiplier} + \text{offset}\)

**8.70.3.9**  
void response_modify_n2s (const Variables & native_vars, const Response & native_response, Response & recast_response, int native_offset, int recast_offset, int num_responses) const

map responses from native to scaled variable space
scaling response mapping: modifies response from a model (user/native) for use in iterators (scaled) – not including multi_objective_modify

**8.70.3.10**  
void response_modify_s2n (const Variables & native_vars, const Response & scaled_response, Response & native_response, int scaled_offset, int native_offset, int num_responses) const

map responses from scaled to native variable space
scaling response mapping: modifies response from scaled (iterator) to native (user) space – not including multi_objective_retrieve

**8.70.3.11**  
RealMatrix lin_coeffs_modify_n2s (const RealMatrix & src_coeffs, const RealVector & cv_multipliers, const RealVector & lin_multipliers) const

compute scaled linear constraint matrix given design variable multipliers and linear scaling multipliers. Only scales components corresponding to continuous variables so for src_coeffs of size MxN, lin_multipliers.size() \(\leq\) M, cv_multipliers.size() \(\leq\) N

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

- DakotaMinimizer.H
- DakotaMinimizer.C
MixedConstraints Class Reference

the default data view (no variable or domain type array merging).

Inheritance diagram for MixedConstraints:

MixedConstraints

Constraints

Public Member Functions

- **MixedConstraints ()**
  default constructor

- **MixedConstraints (const ProblemDescDB &problem_db, const std::pair< short, short > &view, const Sizet2DArray &vars_comps)**
  standard constructor

- **~MixedConstraints ()**
  destructor

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

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

Protected Member Functions

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

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

- **void build_active_views ()**
  construct active views of all variables bounds arrays

- **void build_inactive_views ()**
  construct inactive views of all variables bounds arrays
8.71.1 Detailed Description

the default data view (no variable or domain type array merging).

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the
continuous and discrete domain types. The MixedConstraints derived class separates the design, uncertain, and
state variable types as well as the continuous and discrete domain types. The result is separate lower and upper
bounds arrays for continuous design, discrete design, uncertain, continuous state, and discrete state variables.
This is the default approach, so all iterators and strategies not specifically utilizing the All or Merged views use
this approach (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to
the Constraints constructor in Model).

8.71.2 Constructor & Destructor Documentation

8.71.2.1 MixedConstraints (const ProblemDescDB & problem_db, const std::pair< short, short > &
view, const Sizet2DArray & vars_comps)

standard constructor

In this class, mixed continuous/discrete variables are used. Most iterators/strategies use this approach, which is
the default in Constraints::get_constraints().

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

- MixedConstraints.H
- MixedConstraints.C
8.72 MixedVariables Class Reference

the default data view (no variable or domain type array merging).
Inheritance diagram for MixedVariables::

```
Variables

MixedVariables
```

Public Member Functions

- **MixedVariables ()**
  
  *default constructor*

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

- **~MixedVariables ()**
  
  *destructor*

- **void read (std::istream &s)**
  
  *read a variables object from an std::istream*

- **void write (std::ostream &s) const**
  
  *write a variables object to an std::ostream*

- **void write_aprepro (std::ostream &s) const**
  
  *write a variables object to an std::ostream in aprepro format*

- **void read_tabular (std::istream &s)**

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

Protected Member Functions

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

- **void reshape_rep (const Sizet2DArray &vars_comps)**
  
  *Used by reshape() to reshape the contents of a letter class.*
• void build_active_views()
  
  construct active views of all variables arrays

• void build_inactive_views()
  
  construct inactive views of all variables arrays

### 8.72.1 Detailed Description

the default data view (no variable or domain type array merging).

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MixedVariables derived class separates the design, uncertain, and state variable types as well as the continuous and discrete domain types. The result is separate arrays for continuous design, discrete design, uncertain, continuous state, and discrete state variables. This is the default approach, so all iterators and strategies not specifically utilizing the All or Merged views use this approach (see Variables::get_-variables(problem_db)).

### 8.72.2 Constructor & Destructor Documentation

#### 8.72.2.1 MixedVariables (const ProblemDescDB & problem_db, const std::pair<short, short> & view)

standard constructor

In this class, the distinct approach is used (design, uncertain, and state variable types and continuous and discrete domain types are distinct). Most iterators/strategies use this approach.

### 8.72.3 Member Function Documentation

#### 8.72.3.1 void read_tabular (std::istream & s) [virtual]

Presumes variables object is already appropriately sized to receive!

Reimplemented from Variables.

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

- MixedVariables.H
- MixedVariables.C
8.73 Model Class Reference

Base class for the model class hierarchy.

Inheritance diagram for Model::

```
Model
├── NestedModel
├── RecastModel
├── SingleModel
├── SurrogateModel
│   ├── DataFitSurrModel
│   └── HierarchSurrModel
```

Public Member Functions

- **Model ()**
  
  *default constructor*

- **Model (ProblemDescDB &problem_db)**
  
  *standard constructor for envelope*

- **Model (const Model &model)**
  
  *copy constructor*

- **virtual ~Model ()**
  
  *destructor*

- **Model operator= (const Model &model)**
  
  *assignment operator*

- **virtual Iterator & subordinate_iterator ()**
  
  *return the sub-iterator in nested and surrogate models*

- **virtual Model & subordinate_model ()**
  
  *dive through model recursions that may bypass some components.*

- **virtual Model & surrogate_model ()**
  
  *return the approximation sub-model in surrogate models*

- **virtual Model & truth_model ()**
  
  *return the truth sub-model in surrogate models*

- **virtual void derived_subordinate_models (ModelList &ml, bool recurse_flag)**
  
  *portion of subordinate_models() specific to derived model classes*
• virtual void update_from_subordinate_model (bool recurse_flag=true)
  propagate vars/labels/bounds/targets from the bottom up

• virtual Interface & interface ()
  or NestedModel::optionalInterface

• virtual void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  squares terms

• virtual void surrogate_bypass (bool bypass_flag)
  models contained within this model

• virtual void surrogate_function_indices (const IntSet &surr_fn_indices)
  set the (currently active) surrogate function index set

• virtual void build_approximation ()
  build a new SurrogateModel approximation

• virtual bool build_approximation (const Variables &vars, const Response &response)
  response at vars

• virtual void update_approximation (const Variables &vars, const Response &response, bool rebuild_flag)
  update an existing surrogate model with a new anchor

• virtual void update_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  update an existing surrogate model with new data points

• virtual void append_approximation (const Variables &vars, const Response &response, bool rebuild_flag)
  append a single point to an existing surrogate model’s data

• virtual void append_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  append multiple points to an existing surrogate model’s data

• virtual bool force_rebuild ()
  based on changes in the inactive data

• virtual Array< Approximation > & approximations ()
  retrieve the set of Approximations within a DataFitSurrModel

• virtual const RealVectorArray & approximation_coefficients ()
  within a DataFitSurrModel

• virtual void approximation_coefficients (const RealVectorArray &approx_coeffs)
  a DataFitSurrModel
- virtual void `print_coefficients` (std::ostream &s, size_t index) const
  within a `DataFitSurrModel`

- virtual const RealVector & `approximation_variances` (const RealVector &c_vars)
  `Approximation` within a `DataFitSurrModel`.

- virtual const List<SurrogateDataPoint> & `approximation_data` (size_t index)
  instance within a `DataFitSurrModel`

- virtual void `compute_correction` (const Response &truth_response, const Response &approx_response, const RealVector &c_vars)
  compute correction factors for use in SurrogateModels

- virtual void `auto_correction` (bool correction_flag)
  manages automatic application of correction factors in SurrogateModels

- virtual bool `auto_correction` ()
  model’s responses

- virtual void `apply_correction` (Response &approx_response, const RealVector &c_vars, bool quiet_flag=false)
  apply correction factors to approx_response (for use in SurrogateModels)

- virtual void `component_parallel_mode` (short mode)
  or 2 (SUB_MODEL/ACTUAL_MODEL/HF_MODEL/TRUTH_MODEL).]

- virtual String `local_eval_synchronization` ()
  return derived model synchronization setting

- virtual int `local_eval_concurrency` ()
  return derived model asynchronous evaluation concurrency

- virtual void `serve` ()
  a termination message is received from stop_servers().

- virtual void `stop_servers` ()
  particular model when iteration on the model is complete.

- virtual bool `derived_master_overload` () const
  of trying to run a multiprocessor job on the master.

- virtual void `inactive_view` (short view, bool recurse_flag=true)
  update the Model’s inactive view based on higher level (nested) context

- virtual const String & `interface_id` () const
virtual int evaluation_id () const

Return the current function evaluation id for the Model.

virtual void set_evaluation_reference ()

Set the reference points for the evaluation counters within the Model.

virtual void fine_grained_evaluation_counters ()

Request fine-grained evaluation reporting within the Model.

virtual void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const

Print an evaluation summary for the Model.

ModelList & subordinate_models (bool recurse_flag=true)

return the sub-models in nested and surrogate models

void compute_response ()

Compute the Response at currentVariables (default ActiveSet).

void compute_response (const ActiveSet &set)

Compute the Response at currentVariables (specified ActiveSet).

void asynch_compute_response ()

Response at currentVariables (default ActiveSet).

void asynch_compute_response (const ActiveSet &set)

Response at currentVariables (specified ActiveSet).

const IntResponseMap & synchronize ()

complete set of results from a group of asynchronous evaluations.

const IntResponseMap & synchronize_nowait ()

available results from a group of asynchronous evaluations.

void init_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)

configuration in modelPCIterMap

void init_serial ()

modify some default settings to behave properly in serial.

void set_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)

from modelPCIterMap

void free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
dealocate communicator partitions for a model

- void **stop_configurations**()
  terminate **serve_configurations()** on other iteratorComm processors

- int **serve_configurations**()
  to balance **init_communicators()** calls on iteratorComm rank 0

- void **estimate_message_lengths**()
  estimate messageLengths for a model

- void **assign_rep** (Model *model_rep, bool ref_count_incr=true)
  replaces existing letter with a new one

- size_t **tv** () const
  returns total number of vars

- size_t **cv** () const
  returns number of active continuous variables

- size_t **div** () const
  returns number of active discrete integer vars

- size_t **drv** () const
  returns number of active discrete real vars

- size_t **icv** () const
  returns number of inactive continuous variables

- size_t **idiv** () const
  returns number of inactive discrete integer vars

- size_t **idrv** () const
  returns number of inactive discrete real vars

- size_t **acv** () const
  returns total number of continuous variables

- size_t **adiv** () const
  returns total number of discrete integer vars

- size_t **adriv** () const
  returns total number of discrete real vars

- void **active_variables** (const Variables &vars)
  set the active variables in currentVariables
• const RealVector & continuous_variables () const
  return the active continuous variables from currentVariables

• void continuous_variables (const RealVector &c_vars)
  set the active continuous variables in currentVariables

• void continuous_variable (const Real &c_var, const size_t &i)
  set an active continuous variable in currentVariables

• const IntVector & discrete_int_variables () const
  return the active discrete integer variables from currentVariables

• void discrete_int_variables (const IntVector &d_vars)
  set the active discrete integer variables in currentVariables

• void discrete_int_variable (const int &d_var, const size_t &i)
  set an active discrete integer variable in currentVariables

• const RealVector & discrete_real_variables () const
  return the active discrete real variables from currentVariables

• void discrete_real_variables (const RealVector &d_vars)
  set the active discrete real variables in currentVariables

• void discrete_real_variable (const Real &d_var, const size_t &i)
  set an active discrete real variable in currentVariables

• StringMultiArrayConstView continuous_variable_types () const
  return the active continuous variable types from currentVariables

• StringMultiArrayConstView discrete_int_variable_types () const
  return the active discrete variable types from currentVariables

• StringMultiArrayConstView discrete_real_variable_types () const
  return the active discrete variable types from currentVariables

• UIntMultiArrayConstView continuous_variable_ids () const
  return the active continuous variable identifiers from currentVariables

• const RealVector & inactive_continuous_variables () const
  return the inactive continuous variables in currentVariables

• void inactive_continuous_variables (const RealVector &i_c_vars)
  set the inactive continuous variables in currentVariables
• const IntVector & inactive_discrete_int_variables() const
  return the inactive discrete variables in currentVariables

• void inactive_discrete_int_variables(const IntVector &i_d_vars)
  set the inactive discrete variables in currentVariables

• const RealVector & inactive_discrete_real_variables() const
  return the inactive discrete variables in currentVariables

• void inactive_discrete_real_variables(const RealVector &i_d_vars)
  set the inactive discrete variables in currentVariables

• UIntMultiArrayConstView inactive_continuous_variable_ids() const
  return the inactive continuous variable identifiers from currentVariables

• const RealVector & all_continuous_variables() const
  return all continuous variables in currentVariables

• void all_continuous_variables(const RealVector &a_c_vars)
  set all continuous variables in currentVariables

• void all_continuous_variable(const Real &a_c_var, const size_t &i)
  set a variable within the all continuous variables in currentVariables

• const IntVector & all_discrete_int_variables() const
  return all discrete variables in currentVariables

• void all_discrete_int_variables(const IntVector &a_d_vars)
  set all discrete variables in currentVariables

• void all_discrete_int_variable(const int &a_d_var, const size_t &i)
  set a variable within the all discrete variables in currentVariables

• const RealVector & all_discrete_real_variables() const
  return all discrete variables in currentVariables

• void all_discrete_real_variables(const RealVector &a_d_vars)
  set all discrete variables in currentVariables

• void all_discrete_real_variable(const Real &a_d_var, const size_t &i)
  set a variable within the all discrete variables in currentVariables

• const RealVector & all_discrete_real_variables() const
  return all discrete variables in currentVariables

• void all_discrete_real_variables(const RealVector &a_d_vars)
  set all discrete variables in currentVariables

• void all_discrete_real_variable(const Real &a_d_var, const size_t &i)
  set a variable within the all discrete variables in currentVariables

• StringMultiArrayConstView all_continuous_variable_types() const
  return all continuous variable types from currentVariables

• StringMultiArrayConstView all_discrete_int_variable_types() const
return all discrete variable types from currentVariables

- StringMultiArrayConstView all_discrete_real_variable_types () const
  return all discrete variable types from currentVariables

- UIntMultiArrayConstView all_continuous_variable_ids () const
  return all continuous variable identifiers from currentVariables

- const IntSetArray & discrete_design_set_int_values () const
  design set integer variables

- void discrete_design_set_int_values (const IntSetArray &isa)
  design set integer variables

- const RealSetArray & discrete_design_set_real_values () const
  design set integer variables

- void discrete_design_set_real_values (const RealSetArray &rsa)
  design set integer variables

- const RealVector & normal_means () const
  return the normal uncertain variable means

- void normal_means (const RealVector &n_means)
  set the normal uncertain variable means

- void normal_mean (const Real &n_mean, size_t i)
  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

- void normal_std_deviation (const Real &n_std_dev, size_t i)
  set the normal uncertain variable standard deviations

- const RealVector & normal_lower_bounds () const
  return the normal uncertain variable lower bounds

- void normal_lower_bounds (const RealVector &n_lower_bnds)
  set the normal uncertain variable lower bounds

- void normal_lower_bound (const Real &n_lower_bnd, size_t i)
  set the normal uncertain variable lower bounds
- const RealVector & normal_upper_bounds () const
  return the normal uncertain variable upper bounds

- void normal_upper_bounds (const RealVector &n_upper_bnds)
  set the normal uncertain variable upper bounds

- void normal_upper_bound (const Real &n_upper_bnd, size_t i)
  set the normal uncertain variable upper bounds

- const RealVector & lognormal_means () const
  return the lognormal uncertain variable means

- void lognormal_means (const RealVector &ln_means)
  set the lognormal uncertain variable means

- void lognormal_mean (const Real &ln_mean, size_t i)
  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

- void lognormal_std_deviation (const Real &ln_std_dev, size_t i)
  set the lognormal uncertain variable standard deviations

- const RealVector & lognormal_lambdas () const
  return the lognormal uncertain variable lambdas

- void lognormal_lambdas (const RealVector &ln_lambdas)
  set the lognormal uncertain variable lambdas

- void lognormal_lambda (const Real &ln_lambda, size_t i)
  set the lognormal uncertain variable lambdas

- const RealVector & lognormal_zetas () const
  return the lognormal uncertain variable zetas

- void lognormal_zetas (const RealVector &ln_std_devs)
  set the lognormal uncertain variable zetas

- void lognormal_zeta (const Real &ln_std_dev, size_t i)
  set the lognormal uncertain variable zetas
• 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

• void lognormal_error_factor (const Real &ln_err_fact, size_t i)
  set the lognormal uncertain variable error factors

• const RealVector & lognormal_lower_bounds () const
  return the lognormal uncertain variable lower bounds

• void lognormal_lower_bounds (const RealVector &ln_lower_bnds)
  set the lognormal uncertain variable lower bounds

• void lognormal_lower_bound (const Real &ln_lower_bnd, size_t i)
  set the lognormal uncertain variable lower bounds

• const RealVector & lognormal_upper_bounds () const
  return the lognormal uncertain variable upper bounds

• void lognormal_upper_bounds (const RealVector &ln_upper_bnds)
  set the lognormal uncertain variable upper bounds

• void lognormal_upper_bound (const Real &ln_upper_bnd, size_t i)
  set the lognormal uncertain variable upper bounds

• const RealVector & uniform_lower_bounds () const
  return the uniform uncertain variable lower bounds

• void uniform_lower_bounds (const RealVector &u_lower_bnds)
  set the uniform uncertain variable lower bounds

• void uniform_lower_bound (const Real &u_lower_bnd, size_t i)
  set the uniform uncertain variable lower bounds

• const RealVector & uniform_upper_bounds () const
  return the uniform uncertain variable upper bounds

• void uniform_upper_bounds (const RealVector &u_upper_bnds)
  set the uniform uncertain variable upper bounds

• void uniform_upper_bound (const Real &u_upper_bnd, size_t i)
  set the uniform uncertain variable upper bounds

• const RealVector & loguniform_lower_bounds () const
return the loguniform uncertain variable lower bounds

- void loguniform_lower_bounds (const RealVector &lu_lower_bnds)
  set the loguniform uncertain variable lower bounds

- void loguniform_lower_bound (const Real &lu_lower_bnd, size_t i)
  set the loguniform uncertain variable lower bounds

- const RealVector & loguniform_upper_bounds () const
  return the loguniform uncertain variable upper bounds

- void loguniform_upper_bounds (const RealVector &lu_upper_bnds)
  set the loguniform uncertain variable upper bounds

- void loguniform_upper_bound (const Real &lu_upper_bnd, size_t i)
  set the loguniform uncertain variable upper bounds

- const RealVector & triangular_modes () const
  return the triangular uncertain variable modes

- void triangular_modes (const RealVector &t_modes)
  set the triangular uncertain variable modes

- void triangular_mode (const Real &t_mode, size_t i)
  set the triangular uncertain variable modes

- const RealVector & triangular_lower_bounds () const
  return the triangular uncertain variable lower bounds

- void triangular_lower_bounds (const RealVector &t_lower_bnds)
  set the triangular uncertain variable lower bounds

- void triangular_lower_bound (const Real &t_lower_bnd, size_t i)
  set the triangular uncertain variable lower bounds

- const RealVector & triangular_upper_bounds () const
  return the triangular uncertain variable upper bounds

- void triangular_upper_bounds (const RealVector &t_upper_bnds)
  set the triangular uncertain variable upper bounds

- void triangular_upper_bound (const Real &t_upper_bnd, size_t i)
  set the triangular uncertain variable upper bounds

- const RealVector & exponential_betas () const
  return the exponential uncertain variable beta parameters
- void exponential_betas (const RealVector &e_betas)
  set the exponential uncertain variable beta parameters

- void exponential_beta (const Real &e_beta, size_t i)
  set the exponential uncertain variable beta parameters

- const RealVector & beta_alphas () const
  return the beta uncertain variable alphas

- void beta_alphas (const RealVector &b_alphas)
  set the beta uncertain variable alphas

- void beta_alpha (const Real &b_alpha, size_t i)
  set the beta uncertain variable alphas

- const RealVector & beta_betas () const
  return the beta uncertain variable betas

- void beta_betas (const RealVector &b_betas)
  set the beta uncertain variable betas

- void beta_beta (const Real &b_beta, size_t i)
  set the beta uncertain variable betas

- const RealVector & beta_lower_bounds () const
  return the beta uncertain variable lower bounds

- void beta_lower_bounds (const RealVector &b_lower_bnds)
  set the beta uncertain variable lower bounds

- void beta_lower_bound (const Real &b_lower_bnd, size_t i)
  set the beta uncertain variable lower bounds

- const RealVector & beta_upper_bounds () const
  return the beta uncertain variable upper bounds

- void beta_upper_bounds (const RealVector &b_upper_bnds)
  set the beta uncertain variable upper bounds

- void beta_upper_bound (const Real &b_upper_bnd, size_t i)
  set the beta uncertain variable upper bounds

- const RealVector & gamma_alphas () const
  return the gamma uncertain variable alpha parameters
• **void gamma_alphas** (const RealVector &ga_alphas)
  *set the gamma uncertain variable alpha parameters*

• **void gamma_alpha** (const Real &ga_alpha, size_t i)
  *set the gamma uncertain variable alpha parameters*

• **const RealVector & gamma_betas** () const
  *return the gamma uncertain variable beta parameters*

• **void gamma_betas** (const RealVector &ga_betas)
  *set the gamma uncertain variable beta parameters*

• **void gamma_beta** (const Real &ga_beta, size_t i)
  *set the gamma uncertain variable beta parameters*

• **const RealVector & gumbel_alphas** () const
  *return the gumbel uncertain variable alphas*

• **void gumbel_alphas** (const RealVector &gu_alphas)
  *set the gumbel uncertain variable alphas*

• **void gumbel_alpha** (const Real &gu_alpha, size_t i)
  *set the gumbel uncertain variable alphas*

• **const RealVector & gumbel_betas** () const
  *return the gumbel uncertain variable betas*

• **void gumbel_betas** (const RealVector &gu_betas)
  *set the gumbel uncertain variable betas*

• **void gumbel_beta** (const Real &gu_beta, size_t i)
  *set the gumbel uncertain variable betas*

• **const RealVector & frechet_alphas** () const
  *return the frechet uncertain variable alpha parameters*

• **void frechet_alphas** (const RealVector &f_alphas)
  *set the frechet uncertain variable alpha parameters*

• **void frechet_alpha** (const Real &f_alpha, size_t i)
  *set the frechet uncertain variable alpha parameters*

• **const RealVector & frechet_betas** () const
  *return the frechet uncertain variable beta parameters*

• **void frechet_betas** (const RealVector &f_betas)
set the frechet uncertain variable beta parameters

- void frechet_beta (const Real &f_beta, size_t i) 

set the frechet uncertain variable beta parameters

- const RealVector & weibull_alphas () const 

return the weibull uncertain variable alpha parameters

- void weibull_alphas (const RealVector &w_alphas) 

set the weibull uncertain variable alpha parameters

- void weibull_alpha (const Real &w_alpha, size_t i) 

set the weibull uncertain variable alpha parameters

- const RealVector & weibull_betas () const 

return the weibull uncertain variable beta parameters

- void weibull_betas (const RealVector &w_betas) 

set the weibull uncertain variable beta parameters

- void weibull_beta (const Real &w_beta, size_t i) 

set the weibull uncertain variable beta parameters

- const RealVectorArray & histogram_bin_pairs () const 

return the histogram uncertain bin pairs

- void histogram_bin_pairs (const RealVectorArray &h_bin_pairs) 

set the histogram uncertain bin pairs

- const RealVector & poisson_lambdas () const 

return the poisson uncertain variable lambda parameters

- void poisson_lambdas (const RealVector &p_lambdas) 

set the poisson uncertain variable lambda parameters

- void poisson_lambda (const Real &p_lambda, size_t i) 

set the poisson uncertain variable lambda parameters

- const RealVector & binomial_probabilities_per_trial () const 

return the binomial probabilities per each trial (p)

- void binomial_probabilities_per_trial (const RealVector &probs_per_trial) 

set the binomial probabilities per each trial (p)

- void binomial_probability_per_trial (const Real &prob_per_trial, size_t i) 

set the binomial probabilities per each trial (p)
- const IntVector & binomial_num_trials() const
  return the binomial number of trials (N)

- void binomial_num_trials(const IntVector &num_trials)
  set the binomial number of trials (N)

- void binomial_num_trials(const int &num_trials, size_t i)
  set the binomial number of trials (N)

- const RealVector & negative_binomial_probabilities_per_trial() const
  return the negative binomial probabilities per each trial (p)

- void negative_binomial_probabilities_per_trial(const RealVector &probs_per_trial)
  set the negative binomial probabilities per each trial (p)

- void negative_binomial_probability_per_trial(const Real &prob_per_trial, size_t i)
  set the negative binomial probabilities per each trial (p)

- const IntVector & negative_binomial_num_trials() const
  return the negative binomial number of trials (N)

- void negative_binomial_num_trials(const IntVector &num_trials)
  set the negative binomial number of trials (N)

- void negative_binomial_num_trials(const int &num_trials, size_t i)
  set the negative binomial number of trials (N)

- const RealVector & geometric_probabilities_per_trial() const
  return the geometric probabilities per each trial (p)

- void geometric_probabilities_per_trial(const RealVector &probs_per_trial)
  set the geometric probabilities per each trial (p)

- void geometric_probability_per_trial(const Real &prob_per_trial, size_t i)
  set the geometric probabilities per each trial (p)

- const IntVector & hypergeometric_total_population() const
  return the hypergeometric number in total population

- void hypergeometric_total_population(const IntVector &total_pop)
  set the hypergeometric number in total population

- void hypergeometric_total_population(const int &total_pop, size_t i)
  set the hypergeometric number in total population
• const IntVector & hypergeometric_selected_population () const
  return the hypergeometric number in selected population

• void hypergeometric_selected_population (const IntVector &sel_pop)
  set the hypergeometric number in selected population

• void hypergeometric_selected_population (const int &sel_pop, size_t i)
  set the hypergeometric number in selected population

• const IntVector & hypergeometric_num_drawn () const
  return the hypergeometric number failed

• void hypergeometric_num_drawn (const IntVector &num_drawn)
  set the hypergeometric number in total population

• void hypergeometric_num_drawn (const int &num_drawn, size_t i)
  set the hypergeometric number in total population

• const RealVectorArray & histogram_point_pairs () const
  return the histogram uncertain point pairs

• void histogram_point_pairs (const RealVectorArray &h_pt_pairs)
  set the histogram uncertain point pairs

• const RealVectorArray & interval_probabilities () const
  return the interval basic probability values

• void interval_probabilities (const RealVectorArray &int_probs)
  set the interval basic probability values

• const RealVectorArray & interval_bounds () const
  return the interval bounds

• void interval_bounds (const RealVectorArray &int_bounds)
  set the interval bounds

• const RealSymMatrix & uncertain_correlations () const
  return the uncertain variable correlations

• void uncertain_correlations (const RealSymMatrix &uncertain_corr)
  set the uncertain variable correlations

• const IntSetArray & discrete_state_set_int_values () const
  state set integer variables

• void discrete_state_set_int_values (const IntSetArray &isa)
state set integer variables

• const RealSetArray & discrete_state_set_real_values() const
  state set integer variables

• void discrete_state_set_real_values(const RealSetArray &rsa)
  state set integer variables

• StringMultiArrayConstView continuous_variable_labels() const
  return the active continuous variable labels from currentVariables

• void continuous_variable_labels(StringMultiArrayConstView c_v_labels)
  set the active continuous variable labels in currentVariables

• StringMultiArrayConstView discrete_int_variable_labels() const
  return the active discrete variable labels from currentVariables

• void discrete_int_variable_labels(StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables

• StringMultiArrayConstView discrete_real_variable_labels() const
  return the active discrete variable labels from currentVariables

• void discrete_real_variable_labels(StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables

• StringMultiArrayConstView inactive_continuous_variable_labels() const
  return the inactive continuous variable labels in currentVariables

• void inactive_continuous_variable_labels(StringMultiArrayConstView i_c_v_labels)
  set the inactive continuous variable labels in currentVariables

• StringMultiArrayConstView inactive_discrete_int_variable_labels() const
  return the inactive discrete variable labels in currentVariables

• void inactive_discrete_int_variable_labels(StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables

• StringMultiArrayConstView inactive_discrete_real_variable_labels() const
  return the inactive discrete variable labels in currentVariables

• void inactive_discrete_real_variable_labels(StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables

• StringMultiArrayConstView all_continuous_variable_labels() const
  return all continuous variable labels in currentVariables
void all_continuous_variable_labels (StringMultiArrayConstView a_c_v_labels)  
  set all continuous variable labels in currentVariables

void all_continuous_variable_label (const String &a_c_v_label, const size_t &i)  
  set a label within the all continuous labels in currentVariables

StringMultiArrayConstView all_discrete_int_variable_labels () const  
  return all discrete variable labels in currentVariables

void all_discrete_int_variable_labels (StringMultiArrayConstView a_d_v_labels)  
  set all discrete variable labels in currentVariables

void all_discrete_int_variable_label (const String &a_d_v_label, const size_t &i)  
  set a label within the all discrete labels in currentVariables

StringMultiArrayConstView all_discrete_real_variable_labels () const  
  return all discrete variable labels in currentVariables

void all_discrete_real_variable_labels (StringMultiArrayConstView a_d_v_labels)  
  set all discrete variable labels in currentVariables

void all_discrete_real_variable_label (const String &a_d_v_label, const size_t &i)  
  set a label within the all discrete labels in currentVariables

const StringArray & response_labels () const  
  return the response labels from currentResponse

void response_labels (const StringArray &resp_labels)  
  set the response labels in currentResponse

const RealVector & continuous_lower_bounds () const  
  return the active continuous lower bounds from userDefinedConstraints

void continuous_lower_bounds (const RealVector &c_l_bnds)  
  set the active continuous lower bounds in userDefinedConstraints

const RealVector & continuous_upper_bounds () const  
  return the active continuous upper bounds from userDefinedConstraints

void continuous_upper_bounds (const RealVector &c_u_bnds)  
  set the active continuous upper bounds in userDefinedConstraints

const IntVector & discrete_int_lower_bounds () const  
  return the active discrete lower bounds from userDefinedConstraints
• void **discrete_int_lower_bounds** (const IntVector &d_l_bnds)
  set the active discrete lower bounds in userDefinedConstraints

• const IntVector & **discrete_int_upper_bounds** () const
  return the active discrete upper bounds from userDefinedConstraints

• void **discrete_int_upper_bounds** (const IntVector &d_u_bnds)
  set the active discrete upper bounds in userDefinedConstraints

• const RealVector & **discrete_real_lower_bounds** () const
  return the active discrete lower bounds from userDefinedConstraints

• void **discrete_real_lower_bounds** (const RealVector &d_l_bnds)
  set the active discrete lower bounds in userDefinedConstraints

• const RealVector & **discrete_real_upper_bounds** () const
  return the active discrete upper bounds from userDefinedConstraints

• void **discrete_real_upper_bounds** (const RealVector &d_u_bnds)
  set the active discrete upper bounds in userDefinedConstraints

• const RealVector & **inactive_continuous_lower_bounds** () const
  return the inactive continuous lower bounds in userDefinedConstraints

• void **inactive_continuous_lower_bounds** (const RealVector &i_c_l_bnds)
  set the inactive continuous lower bounds in userDefinedConstraints

• const RealVector & **inactive_continuous_upper_bounds** () const
  return the inactive continuous upper bounds in userDefinedConstraints

• void **inactive_continuous_upper_bounds** (const RealVector &i_c_u_bnds)
  set the inactive continuous upper bounds in userDefinedConstraints

• const IntVector & **inactive_discrete_int_lower_bounds** () const
  return the inactive discrete lower bounds in userDefinedConstraints

• void **inactive_discrete_int_lower_bounds** (const IntVector &i_d_l_bnds)
  set the inactive discrete lower bounds in userDefinedConstraints

• const IntVector & **inactive_discrete_int_upper_bounds** () const
  return the inactive discrete upper bounds in userDefinedConstraints

• void **inactive_discrete_int_upper_bounds** (const IntVector &i_d_u_bnds)
  set the inactive discrete upper bounds in userDefinedConstraints

• const RealVector & **inactive_discrete_real_lower_bounds** () const
  return the inactive discrete lower bounds in userDefinedConstraints

• void **inactive_discrete_real_lower_bounds** (const RealVector &i_d_l_bnds)
  set the inactive discrete lower bounds in userDefinedConstraints

• const RealVector & **inactive_discrete_real_upper_bounds** () const
  return the inactive discrete upper bounds in userDefinedConstraints

• void **inactive_discrete_real_upper_bounds** (const RealVector &i_d_u_bnds)
  set the inactive discrete upper bounds in userDefinedConstraints
return the inactive discrete lower bounds in userDefinedConstraints

- void inactive_discrete_real_lower_bounds (const RealVector &i_d_l_bnds)
  set the inactive discrete lower bounds in userDefinedConstraints

- const RealVector & inactive_discrete_real_upper_bounds () const
  return the inactive discrete upper bounds in userDefinedConstraints

- void inactive_discrete_real_upper_bounds (const RealVector &i_d_u_bnds)
  set the inactive discrete upper bounds in userDefinedConstraints

- const RealVector & all_continuous_lower_bounds () const
  return all continuous lower bounds in userDefinedConstraints

- void all_continuous_lower_bounds (const RealVector &a_c_l_bnds)
  set all continuous lower bounds in userDefinedConstraints

- void all_continuous_lower_bound (const Real &a_c_l_bnd, const size_t &i)
  userDefinedConstraints

- const RealVector & all_continuous_upper_bounds () const
  return all continuous upper bounds in userDefinedConstraints

- void all_continuous_upper_bounds (const RealVector &a_c_u_bnds)
  set all continuous upper bounds in userDefinedConstraints

- void all_continuous_upper_bound (const Real &a_c_u_bnd, const size_t &i)
  userDefinedConstraints

- const IntVector & all_discrete_int_lower_bounds () const
  return all discrete lower bounds in userDefinedConstraints

- void all_discrete_int_lower_bounds (const IntVector &a_d_l_bnds)
  set all discrete lower bounds in userDefinedConstraints

- void all_discrete_int_lower_bound (const int &a_d_l_bnd, const size_t &i)
  userDefinedConstraints

- const IntVector & all_discrete_int_upper_bounds () const
  return all discrete upper bounds in userDefinedConstraints

- void all_discrete_int_upper_bounds (const IntVector &a_d_u_bnds)
  set all discrete upper bounds in userDefinedConstraints

- void all_discrete_int_upper_bound (const int &a_d_u_bnd, const size_t &i)
  userDefinedConstraints
8.73 Model Class Reference

- const RealVector & all_discrete_real_lower_bounds () const
  return all discrete lower bounds in userDefinedConstraints

- void all_discrete_real_lower_bounds (const RealVector &a_d_l_bnds)
  set all discrete lower bounds in userDefinedConstraints

- void all_discrete_real_lower_bound (const Real &a_d_l_bnd, const size_t &i)
  userDefinedConstraints

- const RealVector & all_discrete_real_upper_bounds () const
  return all discrete upper bounds in userDefinedConstraints

- void all_discrete_real_upper_bounds (const RealVector &a_d_u_bnds)
  set all discrete upper bounds in userDefinedConstraints

- void all_discrete_real_upper_bound (const Real &a_d_u_bnd, const size_t &i)
  userDefinedConstraints

- size_t num_linear_ineq_constraints () const
  return the number of linear inequality constraints

- size_t num_linear_eq_constraints () const
  return the number of linear equality constraints

- const RealMatrix & linear_ineq_constraint_coeffs () const
  return the linear inequality constraint coefficients

- void linear_ineq_constraint_coeffs (const RealMatrix &lin_ineq_coeffs)
  set the linear inequality constraint coefficients

- const RealVector & linear_ineq_constraint_lower_bounds () const
  return the linear inequality constraint lower bounds

- void linear_ineq_constraint_lower_bounds (const RealVector &lin_ineq_l_bnds)
  set the linear inequality constraint lower bounds

- const RealVector & linear_ineq_constraint_upper_bounds () const
  return the linear inequality constraint upper bounds

- void linear_ineq_constraint_upper_bounds (const RealVector &lin_ineq_u_bnds)
  set the linear inequality constraint upper bounds

- const RealMatrix & linear_eq_constraint_coeffs () const
  return the linear equality constraint coefficients
• void linear_eq_constraint_coeffs (const RealMatrix &lin_eq_coeffs)
  set the linear equality constraint coefficients

• const RealVector & linear_eq_constraint_targets () const
  return the linear equality constraint targets

• void linear_eq_constraint_targets (const RealVector &lin_eq_targets)
  set the linear equality constraint targets

• size_t num_nonlinear_ineq_constraints () const
  return the number of nonlinear inequality constraints

• size_t num_nonlinear_eq_constraints () const
  return the number of nonlinear equality constraints

• const RealVector & nonlinear_ineq_constraint_lower_bounds () const
  return the nonlinear inequality constraint lower bounds

• void nonlinear_ineq_constraint_lower_bounds (const RealVector &nln_ineq_l_bnds)
  set the nonlinear inequality constraint lower bounds

• const RealVector & nonlinear_ineq_constraint_upper_bounds () const
  return the nonlinear inequality constraint upper bounds

• void nonlinear_ineq_constraint_upper_bounds (const RealVector &nln_ineq_u_bnds)
  set the nonlinear inequality constraint upper bounds

• const RealVector & nonlinear_eq_constraint_targets () const
  return the nonlinear equality constraint targets

• void nonlinear_eq_constraint_targets (const RealVector &nln_eq_targets)
  set the nonlinear equality constraint targets

• const UIntArray & merged_discrete_ids () const
  merged into a continuous array in currentVariables

• const Variables & current_variables () const
  return the current variables (currentVariables)

• const Constraints & user_defined_constraints () const
  return the user-defined constraints (userDefinedConstraints)

• const Response & current_response () const
  return the current response (currentResponse)

• ProblemDescDB & problem_description_db () const
return the problem description database (probDescDB)

- **ParallelLibrary & parallel_library** () const
  
  return the parallel library (parallelLib)

- const **String & model_type** () const
  
  return the model type (modelType)

- const **String & model_id** () const
  
  return the model identifier (idModel)

- size_t **num_functions** () const
  
  return number of functions in currentResponse

- const **String & gradient_type** () const
  
  return the gradient evaluation type (gradType)

- const **String & method_source** () const
  
  return the numerical gradient evaluation method source (methodSrc)

- const **String & interval_type** () const
  
  return the numerical gradient evaluation interval type (intervalType)

- bool **ignore_bounds** () const
  
  option for ignoring bounds when numerically estimating derivatives

- bool **central_hess** () const
  
  option for using old 2nd-order scheme when computing finite-diff Hessian

- const RealVector & **fd_gradient_step_size** () const
  
  return the finite difference gradient step size (fdGradSS)

- const IntList & **gradient_id_analytic** () const
  
  return the mixed gradient analytic IDs (gradIdAnalytic)

- const IntList & **gradient_id_numerical** () const
  
  return the mixed gradient numerical IDs (gradIdNumerical)

- const **String & hessian_type** () const
  
  return the Hessian evaluation type (hessType)

- const **String & quasi_hessian_type** () const
  
  return the Hessian evaluation type (quasiHessType)

- const RealVector & **fd_hessian_by_grad_step_size** () const
  
  return gradient-based finite difference Hessian step size (fdHessByGradSS)
• const RealVector & \texttt{fd_hessian_by_fn_step_size}() const
  return function-based finite difference Hessian step size (fdHessByFnSS)

• const IntList & \texttt{hessian_id_analytic}() const
  return the mixed Hessian analytic IDs (hessIdAnalytic)

• const IntList & \texttt{hessian_id_numerical}() const
  return the mixed Hessian analytic IDs (hessIdNumerical)

• const IntList & \texttt{hessian_id_quasi}() const
  return the mixed Hessian analytic IDs (hessIdQuasi)

• const RealVector & \texttt{primary_response_fn_weights}() const
  squares terms. Used by \texttt{ConcurrentStrategy} for Pareto set optimization.

• void \texttt{supports_estimated_derivatives} (bool sed_flag)
  set whether this model should perform or pass on derivative estimation

• void \texttt{init_comms_bcast_flag} (bool icb_flag)
  set initCommsBcastFlag

• int \texttt{evaluation_capacity}() const
  return the evaluation capacity for use in iterator logic

• int \texttt{derivative_concurrency}() const
  return the gradient concurrency for use in parallel configuration logic

• bool \texttt{asynch_flag}() const
  return the asynchronous evaluation flag (asynchEvalFlag)

• void \texttt{asynch_flag} (const bool flag)
  set the asynchronous evaluation flag (asynchEvalFlag)

• short \texttt{output_level}() const
  return the outputLevel

• void \texttt{output_level} (const short level)
  set the outputLevel

• const IntArray & \texttt{message_lengths}() const
  return the array of MPI packed message buffer lengths (messageLengths)

• void \texttt{parallel_configuration_iterator} (const ParConfigLIter &pc_iter)
  set modelPCIter
• const ParConfigIterator & parallel_configuration_iterator () const
  return modelPCIter

• void auto_graphics (const bool flag)
  the model as opposed to graphics posting at the strategy level).

• bool is_null () const
  function to check modelRep (does this envelope contain a letter)

• Model * model_rep () const
  that are not mapped to the top Model level

• Real FDstep1 (FDhelp *, Real h_mag, size_t i)
  function returning finite-difference step size (affected by bounds)

• Real FDstep2 (FDhelp *, Real h, size_t j)
  by bounds)

Protected Member Functions

• Model (BaseConstructor, ProblemDescDB &problem_db)
  derived class constructors - Coplien, p. 139)

• Model (NoDBBaseConstructor, ParallelLibrary &parallel_lib, const std::pair< short, short > &view, const Sizet2DArray &vars_comps, const ActiveSet &set)
  constructed on the fly

• Model (RecastBaseConstructor, ProblemDescDB &problem_db,ParallelLibrary &parallel_lib)
  constructed on the fly

• virtual void derived_compute_response (const ActiveSet &set)
  portion of compute_response() specific to derived model classes

• virtual void derived_asynch_compute_response (const ActiveSet &set)
  portion of asynch_compute_response() specific to derived model classes

• virtual const IntResponseMap & derived_synchronize ()
  portion of synchronize() specific to derived model classes

• virtual const IntResponseMap & derived_synchronize_nowait ()
  portion of synchronize_nowait() specific to derived model classes

• virtual void derived_init_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  portion of init_communicators() specific to derived model classes
virtual void derived_init_serial()
portion of init_serial() specific to derived model classes

virtual void derived_set_communicators(const int &max_iterator_concurrency, bool recurse_flag=true)
portion of set_communicators() specific to derived model classes

virtual void derived_free_communicators(const int &max_iterator_concurrency, bool recurse_flag=true)
portion of free_communicators() specific to derived model classes

Protected Attributes

Variables currentVariables
function evaluations

size_t numDerivVars
corrections where only the active continuous variables are supported)

Response currentResponse
function evaluations

size_t numFns
the number of functions in currentResponse

Constraints userDefinedConstraints
an iterator at startup.

String modelType
type of model: single, nested, or surrogate

String surrogateType
type of surrogate model: local_*, multipoint_*, global_*, or hierarchical

String gradType
grad type: none,numerical,analytic,mixed

String methodSrc
method source: dakota,vendor

String intervalType
interval type: forward,central

bool ignoreBounds
option to ignore bounds when computing finite differences

bool centralHess
option to use old 2nd-order finite diffs for Hessians

- **RealVector** `fdGradSS`
  
  *relative step sizes for numerical gradients*

- **IntList** `gradIdAnalytic`
  
  *analytic id's for mixed gradients*

- **IntList** `gradIdNumerical`
  
  *numerical id's for mixed gradients*

- **String** `hessType`
  
  *Hess type: none, numerical, quasi, analytic, mixed.*

- **String** `quasiHessType`
  
  *quasi-Hessian type: bfgs, damped_bfgs, sr1*

- **RealVector** `fdHessByGradSS`
  
  *relative step sizes for numerical Hessians estimated with 1st-order grad differences*

- **RealVector** `fdHessByFnSS`
  
  *relative step sizes for numerical Hessians estimated with 2nd-order fn differences*

- **IntList** `hessIdAnalytic`
  
  *analytic id's for mixed Hessians*

- **IntList** `hessIdNumerical`
  
  *numerical id's for mixed Hessians*

- **IntList** `hessIdQuasi`
  
  *quasi id's for mixed Hessians*

- **bool** `supportsEstimDerivs`
  
  *whether model should perform or forward derivative estimation*

- **IntArray** `messageLengths`
  
  *and PRPair*

- **ProblemDescDB** & **probDescDB**
  
  *class member reference to the problem description database*

- **ParallelLibrary** & **parallelLib**
  
  *class member reference to the parallel library*

- **ParConfigLIter** `modelPCIter`
  
  *the ParallelConfiguration node used by this model instance*
- short `componentParallelMode`  
  `(SUB_MODEL/HF_MODEL/TRUTH_MODEL)`

- bool `asynchEvalFlag`  
  flags asynch evaluations (local or distributed)

- short `outputLevel`  
  output verbosity level: `[SILENT,QUIET,NORMAL,VERBOSE,DEBUG]_OUTPUT`

- `IntSetArray discreteDesignSetIntValues`  
  values corresponding to discrete design integer set variable

- `RealSetArray discreteDesignSetRealValues`  
  values corresponding to discrete design real set variable

- `RealVector normalMeans`  
  normal uncertain variable means

- `RealVector normalStdDevs`  
  normal uncertain variable standard deviations

- `RealVector normalLowerBnds`  
  normal uncertain variable lower bounds

- `RealVector normalUpperBnds`  
  normal uncertain variable upper bounds

- `RealVector lognormalMeans`  
  lognormal uncertain variable means

- `RealVector lognormalStdDevs`  
  lognormal uncertain variable standard deviations

- `RealVector lognormalLambdas`  
  lognormal uncertain variable lambdas

- `RealVector lognormalZetas`  
  lognormal uncertain variable zetas

- `RealVector lognormalErrFacts`  
  lognormal uncertain variable error factors

- `RealVector lognormalLowerBnds`  
  lognormal uncertain variable lower bounds
- RealVector `lognormalUpperBnds`  
  lognormal uncertain variable upper bounds

- RealVector `uniformLowerBnds`  
  uniform uncertain variable lower bounds

- RealVector `uniformUpperBnds`  
  uniform uncertain variable upper bounds

- RealVector `loguniformLowerBnds`  
  loguniform uncertain variable lower bounds

- RealVector `loguniformUpperBnds`  
  loguniform uncertain variable upper bounds

- RealVector `triangularModes`  
  triangular uncertain variable modes

- RealVector `triangularLowerBnds`  
  triangular uncertain variable lower bounds

- RealVector `triangularUpperBnds`  
  triangular uncertain variable upper bounds

- RealVector `exponentialBetas`  
  exponential uncertain variable betas

- RealVector `betaAlphas`  
  beta uncertain variable alphas

- RealVector `betaBetas`  
  beta uncertain variable betas

- RealVector `betaLowerBnds`  
  beta uncertain variable lower bounds

- RealVector `betaUpperBnds`  
  beta uncertain variable upper bounds

- RealVector `gammaAlphas`  
  gamma uncertain variable alphas

- RealVector `gammaBetas`  
  gamma uncertain variable betas

- RealVector `gumbelAlphas`
• RealVector `gumbelBetas`
gumbel uncertain variable betas

• RealVector `frechetAlphas`
frechet uncertain variable alphas

• RealVector `frechetBetas`
frechet uncertain variable betas

• RealVector `weibullAlphas`
weibull uncertain variable alphas

• RealVector `weibullBetas`
weibull uncertain variable betas

• RealVector `poissonLambdas`
poisson uncertain variable lambdas

• RealVector `binomialProbPerTrial`
binomial uncertain variable probabilities per trial

• IntVector `binomialNumTrials`
binomial uncertain variable numbers of trials

• RealVector `negBinomialProbPerTrial`
negative binomial uncertain variable probabilities per trial

• IntVector `negBinomialNumTrials`
negative binomial uncertain variable numbers of trials

• RealVector `geometricProbPerTrial`
geometric uncertain variable probabilities per trial

• IntVector `hyperGeomTotalPopulation`
hypergeometric uncertain variable numbers in total population

• IntVector `hyperGeomSelectedPopulation`
hypergeometric uncertain variable numbers in selected population

• IntVector `hyperGeomNumDrawn`
hypergeometric uncertain variable numbers failed in population

• RealVectorArray `histogramBinPairs`
histogram uncertain (x,y) bin pairs (continuous linear histogram)
- **RealVectorArray** `histogramPointPairs`
  \textit{histogram uncertain (x,y) point pairs (discrete histogram)}

- **RealVectorArray** `intervalBasicProbs`
  \textit{basic probability values for interval uncertain variables}

- **RealVectorArray** `intervalBounds`
  \textit{interval lower/upper bounds for interval uncertain variables}

- **RealSymMatrix** `uncertainCorrelations`
  \textit{and correlation coefficients for reliability)

- **IntSetArray** `discreteStateSetIntValues`
  \textit{values corresponding to discrete state integer set variable}

- **RealSetArray** `discreteStateSetRealValues`
  \textit{values corresponding to discrete state real set variable}

- **RealVector** `primaryRespFnWts`
  \textit{multiobjective optimization or weighted least squares}

### Private Member Functions

- **Model** `* get_model (ProblemDescDB &problem_db)`
  \textit{Used by the envelope to instantiate the correct letter class.}

  - **int** `estimate_derivatives (const ShortArray &map_asv, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, const bool asynch_flag)`
    \textit{method_source} in the numerical gradient specification.

- **void** `synchronize_derivatives (const Variables &vars, const IntResponseMap &fd_responses, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set)`
  \textit{objects (fd_grad_responses) into a single response (new_response)}

- **void** `update_response (const Variables &vars, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, Response &initial_map_response, const RealMatrix &new_fn_grads, const RealSymMatrixArray &new_fn_hessians)`
  \textit{overlay results to update a response object}

- **void** `update_quasi_hessians (const Variables &vars, Response &new_response, const ActiveSet &original_set)`
perform quasi-Newton Hessian updates

- bool manage_asv (const ShortArray &asv_in, ShortArray &map_asv_out, ShortArray &fd_grad_asv_out, ShortArray &fd_hess_asv_out, ShortArray &quasi_hess_asv_out)
  Coordinates usage of estimate_derivatives() calls based on asv_in.

Private Attributes

- String idModel
  model identifier string from the input file

- int modelEvalId
  evaluations are assimilated into a single higher level evaluation)

- bool estDerivsFlag
  asynch_compute_response()

- int evaluationCapacity
  capacity for concurrent evaluations supported by the Model

- std::map<int, ParConfigLIter> modelPCIterMap
  level as the lookup key

- bool initCommsBcastFlag
  init_communicators(); set from Strategy::init_iterator()

- bool modelAutoGraphicsFlag
  graphics posting at the strategy level)

- ModelList modelList
  used to collect sub-models for subordinate_models()

- VariablesList varsList
  synchronize().

- List< ShortArray > asvList
  asynch_compute_response() to synchronize()

- List< ActiveSet > setList
  asynch_compute_response() to synchronize()

- BoolList initialMapList
  synchronize_derivatives()

- BoolList dbCaptureList
synchronize_derivatives()

- **ResponseList** dbResponseList
  synchronize_derivatives()

- **RealList** deltaList
  transfers deltas from estimate_derivatives() to synchronize_derivatives()

- **IntIntMap** numFDEvalsMap
  responses into numerical gradients.

- **IntIntMap** rawEvalIdMap
  for rekeying responseMap.

- **RealVectorArray** xPrev
  previous parameter vectors used in computing s for quasi-Newton updates

- **RealMatrix** fnGradsPrev
  previous gradient vectors used in computing y for quasi-Newton updates

- **RealSymMatrixArray** quasiHessians
  quasi-Newton Hessian approximations

- **SizetArray** numQuasiUpdates
  number of quasi-Newton Hessian updates applied

- **IntResponseMap** responseMap
  concatenated form. The similar map in Interface contains raw responses.

- **IntResponseMap** graphicsRespMap
  to sequential input into the graphics

- **Model** * modelRep
  pointer to the letter (initialized only for the envelope)

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

---

**Classes**

- **struct** FDhelp
  possibly adjusted for bounds
8.73.1 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.

8.73.2 Constructor & Destructor Documentation

8.73.2.1 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.

8.73.2.2 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.

8.73.2.3 Model (const Model & model)

copy constructor

Copy constructor manages sharing of modelRep and incrementing of referenceCount.

8.73.2.4 ~Model () [virtual]

destructor

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

8.73.2.5 Model (BaseConstructor, ProblemDescDB & problem_db) [protected]

derived class constructors - Coplien, p. 139)

This constructor builds the base class data for all inherited models. get_model() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_model() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Model).
8.73 Model Class Reference

8.73.2.6 Model (RecastBaseConstructor, ProblemDescDB & problem_db, ParallelLibrary & parallel_lib) [protected]

created on the fly
This constructor also builds the base class data for inherited models. However, it is used for recast models which
are instantiated on the fly. Therefore it only initializes a small subset of attributes. Note that parallel_lib is
managed separately from problem_db since parallel_lib is needed even in cases where problem_db is an empty
envelope (i.e., use of dummy_db in Model(NoDBBaseConstructor) above.

8.73.3 Member Function Documentation

8.73.3.1 Model operator= (const Model & model)

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

8.73.3.2 Iterator & subordinate_iterator () [virtual]

return the sub-iterator in nested and surrogate models
return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this opera-
tion must be performed on the original envelope object.
Reimplemented in DataFitSurrModel, NestedModel, and RecastModel.

8.73.3.3 Model & subordinate_model () [virtual]

dive through model recursions that may bypass some components.
return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this opera-
tion must be performed on the original envelope object.
Reimplemented in NestedModel, RecastModel, and SurrogateModel.

8.73.3.4 Model & surrogate_model () [virtual]

return the approximation sub-model in surrogate models
return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this opera-
tion must be performed on the original envelope object.
Reimplemented in DataFitSurrModel, HierarchSurrModel, and RecastModel.

8.73.3.5 Model & truth_model () [virtual]

return the truth sub-model in surrogate models
return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in DataFitSurrModel, HierarchSurrModel, and RecastModel.

### 8.73.3.6 void update_from_subordinate_model (bool recurse_flag = true) [virtual]

propagate vars/labels/bounds/targets from the bottom up

used only for instantiate-on-the-fly model recursions (all RecastModel instantiations and alternate DataFitSurrModel instantiations). Single, Hierarchical, and Nested Models do not redefine the function since they do not support instantiate-on-the-fly. This means that the recursion will stop as soon as it encounters a Model that was instantiated normally, which is appropriate since ProblemDescDB-constructed Models use top-down information flow and do not require bottom-up updating.

Reimplemented in DataFitSurrModel, and RecastModel.

### 8.73.3.7 Interface & interface () [virtual]

or NestedModel::optionalInterface

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in DataFitSurrModel, NestedModel, RecastModel, and SingleModel.

### 8.73.3.8 String local_eval_synchronization () [virtual]

return derived model synchronization setting

SingleModels and HierarchSurrModels redefine this virtual function. A default value of "synchronous" prevents asynch local operations for:

- NestedModels: a subIterator can support message passing parallelism, but not asynch local.
- DataFitSurrModels: while asynch evals on approximations will work due to some added bookkeeping, avoiding them is preferable.

Reimplemented in RecastModel, and SingleModel.

### 8.73.3.9 int local_eval_concurrency () [virtual]

return derived model asynchronous evaluation concurrency

SingleModels and HierarchSurrModels redefine this virtual function.

Reimplemented in RecastModel, and SingleModel.

### 8.73.3.10 const String & interface_id () const [virtual]

return the interface identifier
return by reference requires use of dummy objects, but is important to allow use of \texttt{assign\_rep()} since this operation must be performed on the original envelope object.

Reimplemented in \texttt{DataFitSurrModel}, \texttt{NestedModel}, \texttt{RecastModel}, and \texttt{SingleModel}.

### 8.73.3.11 ModelList & subordinate\_models (bool recurse\_flag = true)

return the sub-models in nested and surrogate models since modelList is built with list insertions (using envelope copies), these models may not be used for model\texttt{.assign\_rep()} since this operation must be performed on the original envelope object. They may, however, be used for letter-based operations (including \texttt{assign\_rep()} on letter contents such as an interface).

### 8.73.3.12 void init\_communicators (const int & max\_iterator\_concurrency, bool recurse\_flag = true)

configuration in modelPCIterMap

The \texttt{init\_communicators()} and \texttt{derived\_init\_communicators()} functions are structured to avoid performing the messageLengths estimation more than once. \texttt{init\_communicators()} (not virtual) performs the estimation and then forwards the results to \texttt{derived\_init\_communicators} (virtual) which uses the data in different contexts.

### 8.73.3.13 void init\_serial()

modify some default settings to behave properly in serial.

The \texttt{init\_serial()} and \texttt{derived\_init\_serial()} functions are structured to separate base class (common) operations from derived class (specialized) operations.

### 8.73.3.14 void estimate\_message\_lengths()

estimate messageLengths for a model

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

### 8.73.3.15 void assign\_rep (Model * model\_rep, bool ref\_count\_incr = true)

replaces existing letter with a new one

Similar to the assignment operator, the \texttt{assign\_rep()} function decrements referenceCount for the old modelRep and assigns the new modelRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign\_rep is passed a letter object and operator\texttt{=} is passed an envelope object). Letter assignment supports two models as governed by ref\_count\_incr:

- ref\_count\_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.
• ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after `get_model()`: a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

8.73.3.16 int derivative_concurrency () const

return the gradient concurrency for use in parallel configuration logic

This function assumes derivatives with respect to the active continuous variables. Therefore, concurrency with respect to the inactive continuous variables is not captured.

8.73.3.17 Real FDstep1 (FDhelp * fdh, Real h_mag, size_t j)

function returning finite-difference step size (affected by bounds)

Auxiliary function to compute forward or first central-difference step size.

8.73.3.18 Real FDstep2 (FDhelp * fdh, Real h, size_t j)

by bounds)

Auxiliary function to second central-difference step size, honoring bounds.

8.73.3.19 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.

8.73.3.20 int estimate_derivatives (const ShortArray & map_asv, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set, const bool asynch_flag) [private]

method_source) in the numerical gradient specification.

Estimate derivatives by computing finite difference gradients, finite difference Hessians, and/or quasi-Newton Hessians. The total number of finite difference evaluations is returned for use by synchronize() to track response arrays, and it could be used to improve management of max_function_evaluations within the iterators.

! new logic

8.73.3.21 void synchronize_derivatives (const Variables & vars, const IntResponseMap & fd_responses, Response & new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set) [private]

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().

8.73.3.22  void update_response (const Variables & vars, Response & new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set, Response & initial_map_response, const RealMatrix & new_fn_grads, const RealSymMatrixArray & new_fn_hessians)  [private]

overlay results to update a response object

Overlay the initial_map_response with numerically estimated new_fn_grads and new_fn_hessians to populate new_response as governed by asv vectors. Quasi-Newton secant Hessian updates are also performed here, since this is where the gradient data needed for the updates is first consolidated. Convenience function used by estimate_derivatives() for the synchronous case and by synchronize_derivatives() for the asynchronous case.

8.73.3.23  void update_quasi_hessians (const Variables & vars, Response & new_response, const ActiveSet & original_set)  [private]

perform quasi-Newton Hessian updates

quasi-Newton updates are performed for approximating response function Hessians using BFGS or SR1 formulations. These Hessians are supported only for the active continuous variables, and a check is performed on the DVV prior to invoking the function.

8.73.3.24  bool manage_asv (const ShortArray & asv_in, ShortArray & map_asv_out, ShortArray & fd_grad_asv_out, ShortArray & fd_hess_asv_out, ShortArray & quasi_hess_asv_out)  [private]

Coordinates usage of estimate_derivatives() calls based on asv_in.

Splits asv_in total request into map_asv_out, fd_grad_asv_out, fd_hess_asv_out, and quasi_hess_asv_out as governed by the responses specification. If the returned use_est_deriv is true, then these asv outputs are used by estimate_derivatives() for the initial map, finite difference gradient evals, finite difference Hessian evals, and quasi-Hessian updates, respectively. If the returned use_est_deriv is false, then only map_asv_out is used.

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

- DakotaModel.H
- DakotaModel.C
8.74 Model::FDhelp Struct Reference

possibly adjusted for bounds

Public Attributes

- const RealVector * Lb
- const RealVector * Ub
- const RealVector * x0
- int shortstep

8.74.1 Detailed Description

possibly adjusted for bounds

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

- DakotaModel.H
8.75 MPIPackBuffer Class Reference

Class for packing MPI message buffers.

Public Member Functions

- **MPIPackBuffer** (int size_=1024)
  Constructor, which allows the default buffer size to be set.

- **~MPIPackBuffer** ()
  Destructor.

- const char ∗ buf ()
  Returns a pointer to the internal buffer that has been packed.

- int size ()
  The number of bytes of packed data.

- int capacity ()
  The allocated size of Buffer.

- void reset ()
  Resets the buffer index in order to reuse the internal buffer.

- void pack (const int ∗data, const int num=1)
  Pack one or more int’s.

- void pack (const u_int ∗data, const int num=1)
  Pack one or more unsigned int’s.

- void pack (const long ∗data, const int num=1)
  Pack one or more long’s.

- void pack (const u_long ∗data, const int num=1)
  Pack one or more unsigned long’s.

- void pack (const short ∗data, const int num=1)
  Pack one or more short’s.

- void pack (const u_short ∗data, const int num=1)
  Pack one or more unsigned short’s.

- void pack (const char ∗data, const int num=1)
  Pack one or more char’s.
• **void pack**(const u_char *data, const int num=1)
  
  *Pack one or more unsigned char’s.*

• **void pack**(const double *data, const int num=1)
  
  *Pack one or more double’s.*

• **void pack**(const float *data, const int num=1)
  
  *Pack one or more float’s.*

• **void pack**(const bool *data, const int num=1)
  
  *Pack one or more bool’s.*

• **void pack**(const int &data)
  
  *Pack a int.*

• **void pack**(const u_int &data)
  
  *Pack a unsigned int.*

• **void pack**(const long &data)
  
  *Pack a long.*

• **void pack**(const u_long &data)
  
  *Pack a unsigned long.*

• **void pack**(const short &data)
  
  *Pack a short.*

• **void pack**(const u_short &data)
  
  *Pack a unsigned short.*

• **void pack**(const char &data)
  
  *Pack a char.*

• **void pack**(const u_char &data)
  
  *Pack a unsigned char.*

• **void pack**(const double &data)
  
  *Pack a double.*

• **void pack**(const float &data)
  
  *Pack a float.*

• **void pack**(const bool &data)
  
  *Pack a bool.*
Protected Member Functions

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

Protected Attributes

- char * Buffer
  
  The internal buffer for packing.

- int Index
  
  The index into the current buffer.

- int Size
  
  The total size that has been allocated for the buffer.

8.75.1 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
8.76 MPIUnpackBuffer Class Reference

Class for unpacking MPI message buffers.

Public Member Functions

- void setup (char *buf_, int size_, bool flag_=false)  
  Method that does the setup for the constructors.

- MPIUnpackBuffer ()  
  Default constructor.

- MPIUnpackBuffer (int size_)  
  Constructor that specifies the size of the buffer.

- MPIUnpackBuffer (char *buf_, int size_, bool flag_=false)  
  Constructor that sets the internal buffer to the given array.

- ~MPIUnpackBuffer ()  
  Destructor.

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

- const char * buf ()  
  Returns a pointer to the internal buffer.

- int size ()  
  Returns the length of the buffer.

- int curr ()  
  Returns the number of bytes that have been unpacked from the buffer.

- void reset ()  
  Resets the index of the internal buffer.

- void unpack (int *data, const int num=1)  
  Unpack one or more int’s.

- void unpack (u_int *data, const int num=1)  
  Unpack one or more unsigned int’s.

- void unpack (long *data, const int num=1)  
  Unpack one or more long’s.
• void unpack (u_long *data, const int num=1)
  Unpack one or more unsigned long's.

• void unpack (short *data, const int num=1)
  Unpack one or more short's.

• void unpack (u_short *data, const int num=1)
  Unpack one or more unsigned short's.

• void unpack (char *data, const int num=1)
  Unpack one or more char's.

• void unpack (u_char *data, const int num=1)
  Unpack one or more unsigned char's.

• void unpack (double *data, const int num=1)
  Unpack one or more double's.

• void unpack (float *data, const int num=1)
  Unpack one or more float's.

• void unpack (bool *data, const int num=1)
  Unpack one or more bool's.

• void unpack (int &data)
  Unpack a int.

• void unpack (u_int &data)
  Unpack a unsigned int.

• void unpack (long &data)
  Unpack a long.

• void unpack (u_long &data)
  Unpack a unsigned long.

• void unpack (short &data)
  Unpack a short.

• void unpack (u_short &data)
  Unpack a unsigned short.

• void unpack (char &data)
  Unpack a char.

• void unpack (u_char &data)
Unpack a **unsigned char**.

- void **unpack**(double &data)
  
  **Unpack a double.**

- void **unpack**(float &data)
  
  **Unpack a float.**

- void **unpack**(bool &data)
  
  **Unpack a bool.**

**Protected Attributes**

- char * **Buffer**
  
  *The internal buffer for unpacking.*

- int **Index**
  
  *The index into the current buffer.*

- int **Size**
  
  *The total size that has been allocated for the buffer.*

- bool **ownFlag**
  
  *If TRUE, then this class owns the internal buffer.*

### 8.76.1 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
8.77 NCSUOptimizer Class Reference

Wrapper class for the NCSU DIRECT optimization library.

Inheritance diagram for NCSUOptimizer::

```
  NCSUOptimizer
  |      |
  v      v
Minimizer Optimizer

Public Member Functions

- **NCSUOptimizer (Model &model)**
  *standard constructor*

- **NCSUOptimizer (Model &model, const int &max_iter, const int &max_eval, double min_box_size=-1., double vol_box_size=-1., double solution_target=-DBL_MAX)**
  *alternate constructor for instantiations "on the fly"*

- **NCSUOptimizer (NoDBBaseConstructor, Model &model)**
  *alternate constructor for Iterator instantiations by name*

- **NCSUOptimizer (const RealVector &var_l_bnds, const RealVector &var_u_bnds, const int &max_iter, const int &max_eval, double(*user_obj_eval)(const RealVector &x), double min_box_size=-1., double vol_box_size=-1., double solution_target=-DBL_MAX)**
  *alternate constructor for instantiations "on the fly"*

- **~NCSUOptimizer ()**
  *destructor*

- **void find_optimum ()**
  *Redefines the run virtual function for the optimizer branch.*

Private Member Functions

- **void initialize ()**
  *shared code among model-based constructors*
• void check_inputs ()
  
  verify problem respects NCSU DIRECT Fortran limits

Static Private Member Functions

• static int objective_eval (int *n, double c[ ], double l[ ], double u[ ], int point[ ], int *maxI, int *start, int *maxfunc, double fvec[ ], int iidata[ ], int *iisize, double ddata[ ], int *idsize, char cdata[ ], int *icsize)
  DIRECT src (DIRbatch.f).

Private Attributes

• short setUpType
  GaussProcApproximation currently uses the user_functions mode.

• Real minBoxSize
  holds the minimum boxsize

• Real volBoxSize
  hold the minimum volume boxsize

• Real solutionTarget
  holds the solution target minimum to drive towards

• RealVector lowerBounds
  holds variable lower bounds passed in for "user_functions" mode.

• RealVector upperBounds
  holds variable upper bounds passed in for "user_functions" mode.

• double(* userObjectiveEval )(const RealVector &x)
  "user_functions" mode.

Static Private Attributes

• static NCSUOptimizer * ncsudirectInstance
  functions in order to avoid the need for static data
8.77.1 Detailed Description

Wrapper class for the NCSU DIRECT optimization library.

The NCSUOptimizer class provides a wrapper for a Fortran 77 implementation of the DIRECT algorithm developed at North Carolina State University. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows:

8.77.2 Constructor & Destructor Documentation

8.77.2.1 NCSUOptimizer (Model & model)

standard constructor

This is the standard constructor with method specification support.

8.77.2.2 NCSUOptimizer (Model & model, const int & max_iter, const int & max_eval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

8.77.2.3 NCSUOptimizer (NoDBBaseConstructor, Model & model)

alternate constructor for Iterator instantiations by name

This is an alternate constructor for Iterator instantiations by name using a Model but no ProblemDescDB.

8.77.2.4 NCSUOptimizer (const RealVector & var_l_bnds, const RealVector & var_u_bnds, const int & max_iter, const int & max_eval, double(*)(const RealVector &x) user_obj_eval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function pointer.

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

- NCSUOptimizer.H
- NCSUOptimizer.C
8.78 NestedModel Class Reference

execution within every evaluation of the model.

Inheritance diagram for NestedModel::

```
Model
    ↑
NestedModel
```

Public Member Functions

- **NestedModel** (ProblemDescDB &problem_db)
  *constructor*

- **~NestedModel** ()
  *destructor*

Protected Member Functions

- void **derived_compute_response** (const ActiveSet &set)
  *portion of compute_response() specific to NestedModel*

- void **derived_asynch_compute_response** (const ActiveSet &set)
  *portion of asynch_compute_response() specific to NestedModel*

- **Iterator & subordinate_iterator** ()
  *return subIterator*

- **Model & subordinate_model** ()
  *return subModel*

- void **derived_subordinate_models** (ModelList &ml, bool recurse_flag)
  *return subModel*

- **Interface & interface** ()
  *return optionalInterface*

- void **surrogate_bypass** (bool bypass_flag)
  *to the subModel for any lower-level surrogates.*

- void **component_parallel_mode** (short mode)
optionalInterface and subModel

- **bool derived_master_overload() const**
  
  evaluation (forwarded to optionalInterface)

- **void derived_init_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)**
  
  set up optionalInterface and subModel for parallel operations

- **void derived_init_serial()**
  
  set up optionalInterface and subModel for serial operations.

- **void derived_set_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)**
  
  set active parallel configuration within subModel

- **void derived_free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)**
  
  (forwarded to optionalInterface and subModel)

- **void serve()**
  
  stop_servers().

- **void stop_servers()**
  
  optionalInterface when iteration on the NestedModel is complete.

- **const String & interface_id() const**
  
  return the optionalInterface identifier

- **int evaluation_id() const**
  
  Return the current evaluation id for the NestedModel.

- **void set_evaluation_reference()**
  
  (request forwarded to optionalInterface and subModel)

- **void fine_grained_evaluation_counters()**
  
  and subModel

- **void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const**
  
  (request forwarded to optionalInterface and subModel)

### Private Member Functions

- **void resolve_real_variable_mapping(const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)**
  
  for a named real mapping, resolve primary index and secondary target
• void `resolve_integer_variable_mapping` (const `String` &map1, const `String` &map2, size_t curr_index, short &inactive_sm_view)
  for a named integer mapping, resolve primary index and secondary target

• size_t `sm_acv_index_map` (size_t pacvm_index, short sacvm_target)
  offset pacvm_index based on sacvm_target to create mapped_index

• size_t `sm_adiv_index_map` (size_t padivm_index, short sadivm_target)
  offset padivm_index based on sadivm_target to create mapped_index

• size_t `sm_adrv_index_map` (size_t padrvm_index, short sadrvm_target)
  offset padrvm_index based on sadrvm_target to create mapped_index

• size_t `cv_index_map` (size_t cv_index)
  offset cv_index to create index into aggregated primary/secondary arrays

• size_t `div_index_map` (size_t div_index)
  offset div_index to create index into aggregated primary/secondary arrays

• size_t `drv_index_map` (size_t drv_index)
  offset drv_index to create index into aggregated primary/secondary arrays

• void `real_variable_mapping` (const `Real` &r_var, size_t mapped_index, short svm_target)
  insert r_var into appropriate recipient

• void `integer_variable_mapping` (const int &i_var, size_t mapped_index, short svm_target)
  insert i_var into appropriate recipient

• void `set_mapping` (const `ActiveSet` &mapped_set, `ActiveSet` &interface_set, bool &opt_interface_map, `ActiveSet` &sub_iterator_set, bool &sub_iterator_map)
  total model evaluation requirements (mapped_set)

• void `response_mapping` (const `Response` &interface_response, const `Response` &sub_iterator_response, `Response` &mapped_response)
  mappings to create the total response for the model

• void `update_inactive_view` (short new_view, short &view)
  update inactive variables view for subIterator based on new_view

• void `update_inactive_view` (const `String` &type, short &view)
  update inactive variables view for subIterator based on type

• void `update_sub_model` ()
  update subModel with current variable values/bounds/labels
Private Attributes

- int nestedModelEvals
  
  \textit{derived\_asynch\_compute\_response()}

- Iterator subIterator
  
  \textit{the sub-iterator that is executed on every evaluation of this model}

- Model subModel
  
  \textit{the sub-model used in sub-iterator evaluations}

- size_t numSubIterFns
  
  \textit{number of sub-iterator response functions prior to mapping}

- size_t numSubIterMappedIneqCon
  
  \textit{sub-iteration results}

- size_t numSubIterMappedEqCon
  
  \textit{sub-iteration results}

- Interface optionalInterface
  
  \textit{the total model response}

- String optInterfacePointer
  
  \textit{the optional interface pointer from the nested model specification}

- Response optInterfaceResponse
  
  \textit{the response object resulting from optional interface evaluations}

- size_t numOptInterfPrimary
  
  \textit{functions) resulting from optional interface evaluations}

- size_t numOptInterfIneqCon
  
  \textit{interface evaluations}

- size_t numOptInterfEqCon
  
  \textit{interface evaluations}

- SizetArray primaryACVarMapIndices
  
  \textit{replace the subModel variable values.}

- SizetArray primaryADIVarMapIndices
  
  \textit{insertions replace the subModel variable values.}

- SizetArray primaryADRVarMapIndices
  
  \textit{insertions replace the subModel variable values.}
8.78.2.1 **void derived_compute_response (const ActiveSet & set)** [protected, virtual]

portion of compute_response() specific to NestedModel

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

Reimplemented from Model.
8.78 NestedModel Class Reference

8.78.2.2 void derived_asynch_compute_response (const ActiveSet & set) [protected, virtual]

portion of asynch_compute_response() specific to NestedModel
Not currently supported by NestedModels (need to add concurrent iterator support). As a result, derived_-synchronize() and derived_synchronize_nowait() are inactive as well).
Reimplemented from Model.

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

evaluation (forwarded to optionalInterface)
Derived master overload for subModel is handled separately in subModel.compute_response() within sub-
Iterator.run().
Reimplemented from Model.

8.78.2.4 void derived_init_communicators (const int & max_iterator_concurrency, bool recurse_flag = true) [inline, protected, virtual]

set up optionalInterface and subModel for parallel operations
Asynchronous flags need to be initialized for the subModel. In addition, max_iterator_concurrency is the outer
level iterator concurrency, not the subiterator concurrency that subModel will see, and recomputing the message_-lengths on the subModel is probably not a bad idea either. Therefore, recompute everything on subModel using
init_communicators()
Reimplemented from Model.

8.78.2.5 int evaluation_id () const [inline, protected, virtual]

Return the current evaluation id for the NestedModel.
return the top level nested evaluation count. To get the lower level eval count, the subModel must be explicitly
queried. This is consistent with the eval counter definitions in surrogate models.
Reimplemented from Model.

8.78.2.6 void response_mapping (const Response & opt_interface_response, const Response & sub_iterator_response, Response & mapped_response) [private]

mappings to create the total response for the model
In the OUU case,

optionalInterface fns = {f}, {g} (deterministic primary functions, constraints)
subiterator fns = {S} (UQ response statistics)

Problem formulation for mapped functions:
minimize \( (f) + \{W\}|S| \)
subject to \( \{g_l\} \leq \{g\} \leq \{g_u\} \)
\{a_l\} \leq [A]\{S\} \leq \{a_u\}
\{g\} = \{g_t\}
[A]\{S\} = \{a_t\}

where \([W]\) is the primary_mapping_matrix user input (primaryRespCoeffs class attribute), \([A]\) is the secondary_mapping_matrix user input (secondaryRespCoeffs class attribute), \({\{g_l\},\{a_l\}}\) are the top level inequality constraint lower bounds, \({\{g_u\},\{a_u\}}\) are the top level inequality constraint upper bounds, and \({\{g_t\},\{a_t\}}\) are the top level equality constraint targets.

NOTE: optionalInterface/subIterator primary fns (obj/lsq/generic fns) overlap but optionalInterface/subIterator secondary fns (ineq/eq constraints) do not. The \([W]\) matrix can be specified so as to allow

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

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

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

8.78.3 Member Data Documentation

8.78.3.1 Model subModel [private]

the sub-model used in sub-iterator evaluations

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

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

- NestedModel.H
- NestedModel.C
8.79  NIDRProblemDescDB Class Reference

The derived input file database utilizing the new IDR parser.

Inheritance diagram for NIDRProblemDescDB::

```
  ProblemDescDB
   └── NIDRProblemDescDB
```

Public Member Functions

- **NIDRProblemDescDB (ParallelLibrary &parallel_lib)**
  - *constructor*

- **~NIDRProblemDescDB ()**
  - *destructor*

- void **derived_parse_inputs** (const char *dakota_input_file, const char *parser_options)
  - *database using IDR.*

- void **derived_broadcast ()**
  - *and after receiving the DB buffer on other processor ranks*

- void **derived_post_process ()**
  - *perform any additional data post-processing*

- **KWH (iface_Rlit)**
- **KWH (iface_false)**
- **KWH (iface_ilit)**
- **KWH (iface_int)**
- **KWH (iface_lit)**
- **KWH (iface_start)**
- **KWH (iface_stop)**
- **KWH (iface_str)**
- **KWH (iface_str2D)**
- **KWH (iface_strL)**
- **KWH (iface_true)**
- **KWH (method_Li)**
- **KWH (method_Real)**
- **KWH (method_Real01)**
- **KWH (method_RealDL)**
- **KWH (method_RealLlit)**
- **KWH (method_Realp)**
• KWH (method_Realz)
• KWH (method_Ri)
• KWH (method_coliny_ea)
• KWH (method_false)
• KWH (method_ilit2)
• KWH (method_ilit2p)
• KWH (method_int)
• KWH (method_intDL)
• KWH (method_lit)
• KWH (method_lit2)
• KWH (method_litc)
• KWH (method_litf)
• KWH (method_litp)
• KWH (method_litpp)
• KWH (method_litpp_final)
• KWH (method_moga_begin)
• KWH (method_moga_final)
• KWH (method_num_intset)
• KWH (method_num_resplevs)
• KWH (method_pint)
• KWH (method_pintz)
• KWH (method_resplevs)
• KWH (method_resplevs01)
• KWH (method_shint)
• KWH (method_slit)
• KWH (method_slit2)
• KWH (method_soga_begin)
• KWH (method_soga_final)
• KWH (method_start)
• KWH (method_stop)
• KWH (method_str)
• KWH (method_strL)
• KWH (method_true)
• KWH (method_tr_final)
• KWH (method_type)
• KWH (method_ushint)
• KWH (method_ushintL)
• KWH (model_Real)
• KWH (model_RealDL)
• KWH (model_intset)
• KWH (model_lit)
• KWH (model_order)
- **KWH** (model_shint)
- **KWH** (model_slit2)
- **KWH** (model_start)
- **KWH** (model_stop)
- **KWH** (model_str)
- **KWH** (model_strL)
- **KWH** (model_true)
- **KWH** (resp_RealDL)
- **KWH** (resp_RealL)
- **KWH** (resp_false)
- **KWH** (resp_intL)
- **KWH** (resp_lit)
- **KWH** (resp_nnintz)
- **KWH** (resp_start)
- **KWH** (resp_stop)
- **KWH** (resp_str)
- **KWH** (resp_strL)
- **KWH** (resp_true)
- **KWH** (strategy_Real)
- **KWH** (strategy_RealL)
- **KWH** (strategy_int)
- **KWH** (strategy_lit)
- **KWH** (strategy_slit)
- **KWH** (strategy_start)
- **KWH** (strategy_str)
- **KWH** (strategy_strL)
- **KWH** (strategy_true)
- **KWH** (var_RealLb)
- **KWH** (var_RealLd)
- **KWH** (var_RealUb)
- **KWH** (var_caulbl)
- **KWH** (var_ceulbl)
- **KWH** (var_dailbl)
- **KWH** (var_darlbl)
- **KWH** (var_intDL)
- **KWH** (var_intL)
- **KWH** (var_intz)
- **KWH** (var_start)
- **KWH** (var_stop)
- **KWH** (var_str)
- **KWH** (var_strL)
- **KWH** (var_true)
- **KWH** (var_vil)
- **KWH** (var_vrl)
Static Public Member Functions

- static void `Var_boundchk` (DataVariablesRep *)
- static void `Var_boundgen` (DataVariablesRep *)
- static void `Var_iboundchk` (DataVariablesRep *)
- static void `Var_iboundgen` (DataVariablesRep *)
- static void `botch` (const char *fmt, ...)
- static void `check_variables` (List<DataVariables> *)
- static void `check_responses` (List<DataResponses> *)
- static void `make_variable_defaults` (List<DataVariables> *)
- static void `make_response_defaults` (List<DataResponses> *)
- static void `squawk` (const char *fmt, ...)
- static void `warn` (const char *fmt, ...)

Static Public Attributes

- static NIDRProblemDescDB *pDDBInstance
  
  functions in order to avoid the need for static data

- static int `nerr`

Static Private Member Functions

- static void `var_stop1` (void *)

Private Attributes

- List< void * > `VIL`

8.79.1 Detailed Description

The derived input file database utilizing the new IDR parser.

The NIDRProblemDescDB class is derived from ProblemDescDB for use by the NIDR parser in processing DAKOTA input file data. For information on modifying the NIDR input parsing procedures, refer to Dakota/docs/Dev_Spec_Change.dox. For more on the parsing technology, see “Specifying and Reading Program Input with NIDR” by David M. Gay (report SAND2008-2261P, which is available in PDF form as http://www.sandia.gov/~dmgay/nidr08.pdf). Source for the routines declared herein is NIDRProblemDescDB.C, in which most routines are so short that a description seems unnecessary.

8.79.2 Member Function Documentation

8.79.2.1 void derived_parse_inputs (const char * dakota_input_file, const char * parser_options)
  
  [virtual]

database using NIDR.
Parse the input file using the Input Deck Reader (IDR) parsing system. IDR populates the IDRProblemDescDB object with the input file data.

Reimplemented from ProblemDescDB.

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

- NIDRProblemDescDB.H
- NIDRProblemDescDB.C
8.80 NL2Res Struct Reference

Auxiliary information passed to calcr and calcj via ur.

Public Attributes

- Real * r
  
  residual \( r = r(x) \)

- Real * J
  
  Jacobian \( J = J(x) \).

- Real * x
  
  corresponding parameter vector

- int nf
  
  function invocation count for \( r(x) \)

8.80.1 Detailed Description

Auxiliary information passed to calcr and calcj via ur.

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

- NL2SOLLeastSq.C
8.81 NL2SOLLeastSq Class Reference

Wrapper class for the NL2SOL nonlinear least squares library.

Inheritance diagram for NL2SOLLeastSq:

```
  Iterator
   |    
   v    
Minimizer
   |    
   v    
LeastSq
   |    
   v    
NL2SOLLeastSq
```

Public Member Functions

- **NL2SOLLeastSq (Model &model)**
  standard constructor

- **NL2SOLLeastSq (NoDBBaseConstructor, Model &model)**
  alternate constructor

- **~NL2SOLLeastSq ()**
  destructor

- **void minimize_residuals ()**
  for the least squares branch.

Static Private Member Functions

- **static void calcr (int *np, int *pp, Real *x, int *nfp, Real *r, int *ui, void *ur, Vf vf)**
  evaluator function for residual vector

- **static void calcj (int *np, int *pp, Real *x, int *nfp, Real *J, int *ui, void *ur, Vf vf)**
  evaluator function for residual Jacobian

Private Attributes

- **int auxprt**
auxiliary printing bits (see Dakota Ref Manual): sum of 1 = x0prt (print initial guess) 2 = solprt (print final solution) 4 = 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 deltax
  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 J^T 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)
8.81 NL2SOLLeastSq Class Reference

- int rdreq
  
  whether to compute the regression diagnostic vector (regression_diagnostics)

- Real fprec
  
  expected response function precision (function_precision)

- Real lmax0
  
  initial trust-region radius (initial_trust_radius)

Static Private Attributes

- static NL2SOLLeastSq * nl2solInstance
  
  evaluator functions

8.81.1 Detailed Description

Wrapper class for the NL2OL nonlinear least squares library.

The NL2SOLLeastSq class provides a wrapper for NL2OL (TOMS Algorithm 573), in the updated form of Port Library routines dnfgb from Bell Labs; see http://www.netlib.org/port/readme. The Fortran from Port has been turned into C by f2c. NL2OL uses a function pointer approach for which passed functions must be either global functions or static member functions.

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

- NL2SOLLeastSq.H
- NL2SOLLeastSq.C
8.82 NLPQLPOptimizer Class Reference

Wrapper class for the NLPQLP optimization library, Version 2.0.

Inheritance diagram for NLPQLPOptimizer:

Iterate

Minimizer

Optimizer

NLPQLPOptimizer

Public Member Functions

- **NLPQLPOptimizer (Model &model)**
  
  *standard constructor*

- **NLPQLPOptimizer (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor*

- **~NLPQLPOptimizer ()**
  
  *destructor*

- **void find_optimum ()**
  
  *Redefines the run virtual function for the optimizer branch.*

Protected Member Functions

- **void derived_initialize_run ()**
  
  *performs run-time set up*

Private Member Functions

- **void initialize ()**
  
  *Shared constructor code.*

- **void allocate_workspace ()**
  
  *Allocates workspace for the optimizer.*
• void deallocate_workspace ()
  Releases workspace memory.

• void allocate_constraints ()
  Allocates constraint mappings.

Private Attributes

• int L
  the serial version by setting L=1.

• int numEqConstraints
  numEqConstraints : Number of equality constraints.

• int MMAX
  MMAX must be at least one and greater or equal to M.

• int N
  N : Number of optimization variables.

• int NMAX
  than N.

• int MNN2
  MNN2 : Must be equal to M+N+N+2.

• double * X
  function values should be computed simultaneously.

• double * F
  values to be computed from L iterates stored in X.

• double * G
  function values to be computed from L iterates stored in X.

• double * DF
  of F to compute DF.

• double * DG
  has to be equal to MMAX.

• double * U
  inequality constraints should be nonnegative.
• double * C
  to NMAX.

• double * D
  array D.

• double ACC
  than the accuracy by which gradients are computed.

• double ACCQP
  by NLPQLP and subsequently multiplied by 1.0D+4.

• double STPMIN
  by STPMIN**(1/L-1). If STPMIN<=0, then STPMIN=ACC is used.

• int MAXFUN
  than 50.

• int MAXIT
  gradients (e.g. 100).

• int MAX_NM
  MAX_NM=0, monotone line search is performed.

• double TOL_NM
  non-negative (e.g. 0.1).

• int IPRINT
  values are displayed during the line search.

• int MODE
  function in C and D in form of an LDL decomposition.

• int IOUT
  write-statements start with 'WRITE(IOUT,... '.

• int IFAIL
  constraint.

• double * WA
  WA(LWA) : WA is a real working array of length LWA.

• int LWA
  LWA : LWA value extracted from NLPQLP20.f.

• int * KWA
8.82 NLPQLP Optimizer Class Reference

\[ KWA(LKWA) : The user has to provide working space for an integer array. \]

- int LKWA
  \[ LKWA : LKWA should be at least N+10. \]

- int * ACTIVE
  \[ ACTIVE(J)=.TRUE., J=1,\ldots,M. \]

- int LACTIVE
  \[ least 2*M+10. \]

- int LQL
  \[ contains only an upper triangular factor. \]

- int numNlpqlConstr
  \[ total number of constraints seen by NLPQL \]

- SizetList nonlinIneqConMappingIndices
  \[ constraints used in computing the corresponding NLPQL constraints. \]

- RealList nonlinIneqConMappingMultipliers
  \[ constraints to the corresponding NLPQL constraints. \]

- RealList nonlinIneqConMappingOffsets
  \[ constraints to the corresponding NLPQL constraints. \]

- SizetList linIneqConMappingIndices
  \[ constraints used in computing the corresponding NLPQL constraints. \]

- RealList linIneqConMappingMultipliers
  \[ constraints to the corresponding NLPQL constraints. \]

- RealList linIneqConMappingOffsets
  \[ constraints to the corresponding NLPQL constraints. \]

### 8.82.1 Detailed Description

Wrapper class for the NLPQLP optimization library, Version 2.0.

**************************************************************

AN IMPLEMENTATION OF A SEQUENTIAL QUADRATIC PROGRAMMING METHOD FOR SOLVING NONLINEAR OPTIMIZATION PROBLEMS BY DISTRIBUTED COMPUTING AND NON-MONOTONE LINE SEARCH

This subroutine solves the general nonlinear programming problem
minimize \( F(X) \) subject to \( G(J,X) = 0 \), \( J=0,\ldots,\text{ME} \), \( G(J,X) \geq 0 \), \( J=\text{ME}+1,\ldots,\text{M} \), \( X_L \leq X \leq X_U \)

and is an extension of the code NLPQLD. NLPQLP is specifically tuned to run under distributed systems. A new input parameter \( L \) is introduced for the number of parallel computers, that is the number of function calls to be executed simultaneously. In case of \( L=1 \), NLPQLP is identical to NLPQLD. Otherwise the line search is modified to allow \( L \) parallel function calls in advance. Moreover the user has the opportunity to used distributed function calls for evaluating gradients.

The algorithm is a modification of the method of Wilson, Han, and Powell. In each iteration step, a linearly constrained quadratic programming problem is formulated by approximating the Lagrangian function quadratically and by linearizing the constraints. Subsequently, a one-dimensional line search is performed with respect to an augmented Lagrangian merit function to obtain a new iterate. Also the modified line search algorithm guarantees convergence under the same assumptions as before.

For the new version, a non-monotone line search is implemented which allows to increase the merit function in case of instabilities, for example caused by round-off errors, errors in gradient approximations, etc.

The subroutine contains the option to predetermine initial guesses for the multipliers or the Hessian of the Lagrangian function and is called by reverse communication.

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

- NLPQLPOptimizer.H
- NLPQLPOptimizer.C
8.83 NLSSOLLeastSq Class Reference

Wrapper class for the NLSSOL nonlinear least squares library.

Inheritance diagram for NLSSOLLeastSq:

```
  Iterator
   |    
  Minimizer
   |    
LeastSq   SOLBase
   |  
NLSSOLLeastSq
```

Public Member Functions

- **NLSSOLLeastSq (Model &model)**
  
  *standard constructor*

- **NLSSOLLeastSq (NoDBBaseConstructor, Model &model)**

  *alternate constructor*

- **~NLSSOLLeastSq ()**

  *destructor*

- **void minimize_residuals ()**

  *for the least squares branch.*

Static Private Member Functions

- **static void least_sq_eval (int &mode, int &m, int &n, int &nrowfj, double *x, double *f, double *gradf, int &nstate)**

  *least squares terms (passed by function pointer to NLSSOL).*

Static Private Attributes

- **static NLSSOLLeastSq * nlssolInstance**

  *functions in order to avoid the need for static data*
8.83.1 Detailed Description

Wrapper class for the NLSSOL nonlinear least squares library.

The NLSSOLLeastSq class provides a wrapper for NLSSOL, a Fortran 77 sequential quadratic programming library from Stanford University marketed by Stanford Business Associates. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any nonstatic attribute used within static member functions must be either local to that function or accessed through a static pointer.

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

8.83.2 Constructor & Destructor Documentation

8.83.2.1 NLSSOLLeastSq (Model & model)

standard constructor

This is the primary constructor. It accepts a Model reference.

8.83.2.2 NLSSOLLeastSq (NoDBBaseConstructor, Model & model)

alternate constructor

This is an alternate constructor which accepts a Model but does not have a supporting method specification from the ProblemDescDB.

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

- NLSSOLLeastSq.H
- NLSSOLLeastSq.C
8.84 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.

8.84.1 Detailed Description

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

NoDBBaseConstructor is used to overload the constructor used for on-the-fly instantiations in which Problem.DescDB queries cannot be used. Putting this struct here avoids circular dependencies.

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

- global_defs.h
8.85 NonD Class Reference

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

Inheritance diagram for NonD:

![Inheritance Diagram]

### Public Member Functions

- void `initialize_random_variables` (short u_space_type)
  
  *initialize natafTransform based on distribution data from iteratedModel*

- void `initialize_random_variables` (const Pecos::ProbabilityTransformation &transform)
  
  *alternate form: initialize natafTransform based on incoming data*

- void `requested_levels` (const RealVectorArray &reqResp_levels, const RealVectorArray &reqProb_levels, const RealVectorArray &reqRel_levels, const RealVectorArray &reqGenRel_levels, short resp_lev_target, bool cdf_flag)
  
  *combination with alternate ctors*

- void `moments` (const RealVector &means, const RealVector &std_devs)
  
  *set meanStats and stdDevStats*

- void `distribution_parameter_derivatives` (bool dist_param_derivs)
  
  *set distParamDerivs*

### Protected Member Functions

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

- **NonD** (NoDBBaseConstructor, Model &model)
  
  *alternate constructor for sample generation and evaluation “on the fly”*

- **NonD** (NoDBBaseConstructor, const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  *alternate constructor for sample generation “on the fly”*

- **~NonD** ()
  
  *destructor*
void derived_initialize_run ()

portions of initialize_run specific to derived iterators

void run ()

and may contain pre/post steps in lieu of separate pre/post

void derived_finalize_run ()

portions of finalize_run specific to derived iterators

const Response & response_results () const

return the final statistics from the nondeterministic iteration

void response_results_active_set (const ActiveSet &set)

set the active set within finalStatistics

virtual void quantify_uncertainty ()=0

distributions into response statistics

virtual void initialize_final_statistics ()

initializes finalStatistics for storing NonD final results

void initialize_random_variable_types (short u_space_type)

initializes ranVarTypesX and ranVarTypesU within natafTransform

void initialize_random_variable_parameters ()

ranVarUpperBndsX, and ranVarAddtlParamsX within natafTransform

void initialize_final_statistics_gradients ()

initializes finalStatistics::functionGradients

Static Protected Member Functions

• static void vars_u_to_x_mapping (const Variables &u_vars, Variables &xVars)

from NonD Iterators to x-space variables for Model evaluations.

• static void set_u_to_x_mapping (const ActiveSet &u_set, ActiveSet &x_set)

from NonD Iterators to x-space ActiveSets for Model evaluations.

• static void resp_x_to_u_mapping (const Variables &x_vars, const Variables &u_vars, const Response &x_response, Response &u_response)

Model evaluations to u-space responses for return to NonD Iterators.
Protected Attributes

- **NonD * prevNondInstance**
  
  *pointer containing previous value of nondInstance*

- **Pecos::ProbabilityTransformation natafTransform**
  
  *data for performing transformations from X -> Z -> U and back.*

- **size_t numContDesVars**
  
  *distribution for All view modes*

- **size_t numDiscIntDesVars**
  
  *histogram distributions for All view modes*

- **size_t numDiscRealDesVars**
  
  *histogram distributions for All view modes*

- **size_t numDesignVars**
  
  *total number of design variables*

- **size_t numContStateVars**
  
  *distribution for All view modes*

- **size_t numDiscIntStateVars**
  
  *histogram distributions for All view modes*

- **size_t numDiscRealStateVars**
  
  *histogram distributions for All view modes*

- **size_t numStateVars**
  
  *total number of state variables*

- **size_t numNormalVars**
  
  *number of normal uncertain variables (native space)*

- **size_t numLognormalVars**
  
  *number of lognormal uncertain variables (native space)*

- **size_t numUniformVars**
  
  *number of uniform uncertain variables (native space)*

- **size_t numLoguniformVars**
  
  *number of loguniform uncertain variables (native space)*

- **size_t numTriangularVars**
  
  *number of triangular uncertain variables (native space)*
• `size_t numExponentialVars`
  number of exponential uncertain variables (native space)

• `size_t numBetaVars`
  number of beta uncertain variables (native space)

• `size_t numGammaVars`
  number of gamma uncertain variables (native space)

• `size_t numGumbelVars`
  number of gumbel uncertain variables (native space)

• `size_t numFrechetVars`
  number of frechet uncertain variables (native space)

• `size_t numWeibullVars`
  number of weibull uncertain variables (native space)

• `size_t numHistogramBinVars`
  number of histogram bin uncertain variables (native space)

• `size_t numPoissonVars`
  number of Poisson uncertain variables (native space)

• `size_t numBinomialVars`
  number of binomial uncertain variables (native space)

• `size_t numNegBinomialVars`
  number of negative binomial uncertain variables (native space)

• `size_t numGeometricVars`
  number of geometric uncertain variables (native space)

• `size_t numHyperGeomVars`
  number of hypergeometric uncertain variables (native space)

• `size_t numHistogramPtVars`
  number of histogram point uncertain variables (native space)

• `size_t numIntervalVars`
  number of interval uncertain variables (native space)

• `size_t numContAleatUncVars`
  total number of aleatory uncertain variables (native space)
• size_t numDiscIntAleatUncVars
  total number of aleatory uncertain variables (native space)

• size_t numDiscRealAleatUncVars
  total number of aleatory uncertain variables (native space)

• size_t numAleatoryUncVars
  total number of aleatory uncertain variables (native space)

• size_t numContEpistUncVars
  total number of epistemic uncertain variables (native space)

• size_t numDiscIntEpistUncVars
  total number of epistemic uncertain variables (native space)

• size_t numDiscRealEpistUncVars
  total number of epistemic uncertain variables (native space)

• size_t numEpistemicUncVars
  total number of epistemic uncertain variables (native space)

• size_t numUncertainVars
  total number of uncertain variables (native space)

• size_t numResponseFunctions
  number of response functions

• RealVector meanStats
  means of response functions (calculated in compute_statistics())

• RealVector stdDevStats
  std deviations of response functions (calculated in compute_statistics())

• RealVectorArray requestedRespLevels
  requested response levels for all response functions

• RealVectorArray computedProbLevels
  from requestedRespLevels

• RealVectorArray computedRelLevels
  from requestedRespLevels

• RealVectorArray computedGenRelLevels
  resulting from requestedRespLevels

• short respLevelTarget
or \( z \rightarrow \beta \) (GEN_RELIABILITIES)

- **RealVectorArray requestedProbLevels**
  
  requested probability levels for all response functions

- **RealVectorArray requestedRelLevels**
  
  requested reliability levels for all response functions

- **RealVectorArray requestedGenRelLevels**
  
  requested generalized reliability levels for all response functions

- **RealVectorArray computedRespLevels**
  
  requestedProbLevels, requestedRelLevels, or requestedGenRelLevels

- **size_t totalLevelRequests**
  
  requestedProbLevels, and requestedRelLevels

- **bool cdfFlag**
  
  cumulative/CDF (true) or complementary/CCDF (false)

- **Response finalStatistics**
  
  response means, standard deviations, and probabilities of failure

**Static Protected Attributes**

- **static NonD * nondInstance**
  
  functions in order to avoid the need for static data

**Private Member Functions**

- **void distribute_levels (RealVectorArray &levels)**
  
  response functions if a short-hand specification is employed.

**Private Attributes**

- **bool distParamDerivs**
  
  to standard random variables \( u \) using the chain rule \( df/dx \ dx/du \).
8.85.1 Detailed Description

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

The base class for nondeterministic iterators consolidates uncertain variable data and probabilistic utilities for inherited classes.

8.85.2 Member Function Documentation

8.85.2.1 void initialize_random_variables (short u_space_type)

initialize natafTransform based on distribution data from iteratedModel

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

8.85.2.2 void initialize_random_variables (const Pecos::ProbabilityTransformation & transform)

alternate form: initialize natafTransform based on incoming data

This function is commonly used to publish transformation data when the Model variables are in a transformed space (e.g., u-space) and ProbabilityTransformation::ranVarTypes et al. may not be generated directly. This allows for the use of inverse transformations to return the transformed space variables to their original states.

8.85.2.3 void derived_initialize_run () [inline, protected, virtual]

portions of initialize_run specific to derived iterators

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of initialize_run(). Redefinition by derived classes is optional.

Reimplemented from Iterator.

8.85.2.4 void run () [inline, protected, virtual]

and may contain pre/post steps in lieu of separate pre/post

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

8.85.2.5 void derived_finalize_run () [inline, protected, virtual]

portions of finalize_run specific to derived iterators

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of finalize_run(). Redefinition by derived classes is optional.

Reimplemented from Iterator.
8.85.2.6  void initialize_final_statistics ()  [protected, virtual]

initializes finalStatistics for storing NonD final results
Default definition of virtual function (used by sampling, reliability, and polynomial chaos) defines the set of statistical results to include means, standard deviations, and level mappings.
Reimplemented in NonDInterval.

8.85.2.7  void initialize_random_variable_types (short u_space_type)  [protected]

initializes ranVarTypesX and ranVarTypesU within natafTransform
Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

8.85.2.8  void initialize_random_variable_parameters ()  [protected]

ranVarUpperBndsX, and ranVarAddtlParamsX within natafTransform
Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

8.85.2.9  void vars_u_to_x_mapping (const Variables & u_vars, Variables & x_vars)  [static, protected]

from NonD Iterators to x-space variables for Model evaluations.
Map the variables from iterator space (u) to simulation space (x).

8.85.2.10 void set_u_to_x_mapping (const ActiveSet & u_set, ActiveSet & x_set)  [static, protected]

from NonD Iterators to x-space ActiveSets for Model evaluations.
Define the DVV for x-space derivative evaluations by augmenting the iterator requests to account for correlations.
The documentation for this class was generated from the following files:

- DakotaNonD.H
- DakotaNonD.C
8.86 NonDAdaptImpSampling Class Reference

Class for the Adaptive Importance Sampling methods within DAKOTA.

Inheritance diagram for NonDAdaptImpSampling:

```
  Iterator
  
  Analyzer
  
  NonD
  
  NonDSampling
  
  NonDAdaptImpSampling
```

Public Member Functions

- `NonDAdaptImpSampling (Model &model)`
  
  constructors standard constructor

- `NonDAdaptImpSampling (Model &model, int samples, int seed, const String &rng, short sampling_type, bool cdf_flag, bool x_space_data, bool x_space_model, bool bounded_model)`

- `~NonDAdaptImpSampling ()`
  
  destructor

- `void quantify_uncertainty ()`
  
  failure.

- `void initialize (const RealVectorArray &initial_points, int resp_fn, const Real &initial_prob, const Real &failure_threshold)`
  
  initial probability to refine, and flags to control transformations

- `void initialize (const RealVector &initial_point, int resp_fn, const Real &initial_prob, const Real &failure_threshold)`
  
  initial probability to refine, and flags to control transformations

- `const Real & get_probability ()`
  
  returns the probability calculated by the importance sampling
Private Member Functions

- void converge_cov ()
  until coefficient of variation converges

- void converge_probability ()
  until probability converges

- void select_init_rep_points (const RealVectorArray &samples)
  select representative points from initial set of samples

- void select_rep_points (const RealVectorArray &samples)
  select representative points from a set of samples

- void calculate_rep_weights ()
  calculate relative weights of representative points

- void generate_samples (RealVectorArray &samples)
  generate a set of samples based on multimodal sampling density

- void calculate_statistics (const RealVectorArray &samples, const size_t &total_sample_number, Real &probability_sum, Real &probability, bool cov_flag, Real &variance_sum, Real &coeff_of_variation)
  the coefficient of variation (if requested)

Private Attributes

- short importanceSamplingType
  integration type (is, ais, mmais) provided by input specification

- bool invertProb
  flag for inversion of probability values using 1.-p

- size_t numRepPoints
  the number of representative points around which to sample

- size_t respFn
  the response function in the model to be sampled

- RealVectorArray initPoints
  the original set of samples passed into the MMAIS routine

- RealVectorArray repPoints
  the set of representative points around which to sample

- RealVector repWeights
the weight associated with each representative point

- RealVector designPoint
  design point at which uncertain space is being sampled

- bool transInitPoints
  initial points

- bool transPoints
  before evaluation

- bool useModelBounds
  flag to control if the sampler should respect the model bounds

- bool initLHS
  flag to identify if initial points are generated from an LHS sample

- Real initProb
  the initial probability (from FORM or SORM)

- Real finalProb
  the final calculated probability (p)

- Real failThresh
  the failure threshold (z-bar) for the problem.

### 8.86.1 Detailed Description

Class for the Adaptive Importance Sampling methods within DAKOTA.

The NonDAdaptImpSampling implements the multi-modal adaptive importance sampling used for reliability calculations. (eventually we will want to broaden this). Need to add more detail to this description.

### 8.86.2 Constructor & Destructor Documentation

#### 8.86.2.1 NonDAdaptImpSampling (Model & model)

constructors standard constructor

This is the primary constructor. It accepts a Model reference.

#### 8.86.2.2 NonDAdaptImpSampling (Model & model, int samples, int seed, const String & rng, short sampling_type, bool cdf_flag, bool x_space_data, bool x_space_model, bool bounded_model)

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.
8.86.3  Member Function Documentation

8.86.3.1  void initialize (const RealVectorArray & initial_points, int resp_fn, const Real & initial_prob, const Real & failure_threshold)

initial probability to refine, and flags to control transformations
Initializes data using a set of starting points.

8.86.3.2  void initialize (const RealVector & initial_point, int resp_fn, const Real & initial_prob, const Real & failure_threshold)

initial probability to refine, and flags to control transformations
Initializes data using only one starting point.

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

- NonDAdaptImpSampling.H
- NonDAdaptImpSampling.C
8.87 NonDBayesCal Class Reference

Generates posterior distribution on model parameters given experiment data.

Inheritance diagram for NonDBayesCal::

```
NonDBayesCal
  NonD
    Analyzer
      Iterator
```

Public Member Functions

- `NonDBayesCal (Model &model)`
  *standard constructor*

- `~NonDBayesCal ()`
  *destructor*

Protected Member Functions

- `void quantify_uncertainty ()`
  *additional variables to be specified here.*

- `void print_results (std::ostream &s)`
  *print the final statistics*

Private Attributes

- `Iterator lhsSampler`
  *LHS sampling iterator.*

- `const int seedSpec`
  *the user seed specification (default is 0)*

- `int numSamples`
  *the current number of samples to evaluate*
8.87 NonDBayesCal Class Reference

- String rngName
  
  *name of the random number generator*

### 8.87.1 Detailed Description

Generates posterior distribution on model parameters given experiment data.

This class provides a wrapper for the functionality provided in the Los Alamos National Laboratory code called GPM/SA (Gaussian Process Models for Simulation Analysis). Although this is a code that provides input/output mapping, it DOES NOT provide the mapping that we usually think of in the NonDeterministic class hierarchy in DAKOTA, where uncertainty in parameter inputs are mapped to uncertainty in simulation responses. Instead, this class takes a pre-existing set of simulation data as well as experimental data, and maps priors on input parameters to posterior distributions on those input parameters, according to a likelihood function. The goal of the MCMC sampling is to produce posterior values of parameter estimates which will produce simulation response values that "match well" to the experimental data. The MCMC is an integral part of the calibration. The data structures in GPM/SA are fairly detailed and nested. Part of this prototyping exercise is to determine what data structures need to be specified and initialized in DAKOTA and sent to GPM/SA, and what data structures will be returned.

### 8.87.2 Constructor & Destructor Documentation

#### 8.87.2.1 NonDBayesCal (Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

### 8.87.3 Member Function Documentation

#### 8.87.3.1 void quantify_uncertainty () [protected, virtual]

additional variables to be specified here.

This method does all the pre-processing necessary to call the GPM/SA code, including running LHS on the model to generate the initial samples, doing some normalization, calling the GPM/SA functions, and returning the posterior parameter distributions.

Implements NonD.

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

- NonDBayesCal.H
- NonDBayesCal.C
8.88 NonDExpansion Class Reference

collocation (SC)

Inheritance diagram for NonDExpansion::

```
NonDExpansion
   |       |
   |       |       |
   |       |       |       |
   |       |       |       |
   |       |       |       |  Iterator
   |       |       |       |
   |       |       |       |  Analyzer
   |       |       |       |
   |       |       |       |  NonD
   |       |       |       |
   |       |       |       |  NonDExpansion
   |       |       |       |
NonDPolynomialChaos  NonDStochCollocation
```

Public Member Functions

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

- **~NonDExpansion ()**
  destructor

- **void print_results (std::ostream &s)**
  print the final statistics

Protected Member Functions

- **virtual void initialize_expansion ()**
  initialize random variable definitions and final stats arrays

- **void construct_g_u_model (Model &g_u_model)**
  recast iteratedModel from x-space to u-space to create g_u_model

- **void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, const UShortArray &quad_order)**
  assign a NonDQuadrature instance within u_space_sampler

- **void construct_sparse_grid (Iterator &u_space_sampler, Model &g_u_model, unsigned short ssg_level, const RealVector &ssg_dim_pref)**
  assign a NonDSparsegrid instance within u_space_sampler
• void `construct_lhs` (Iterator &u_space_sampler, Model &g_u_model)
  assign a NonDLHSSampling instance within u_space_sampler

• void `initialize_u_space_model` ()
  initialize uSpaceModel polynomial approximations with PCE/SC data

• void `construct_expansion_sampler` ()
  construct the expansionSampler operating on uSpaceModel

• void `compute_expansion` ()
  form the expansion by calling uSpaceModel.build_approximation()

• void `compute_statistics` ()
  calculate analytic and numerical statistics from the expansion

• void `update_final_statistics` ()
  update finalStatistics

**Protected Attributes**

• Model uSpaceModel
  u-space recasting and orthogonal polynomial data fit recursions

• short expansionCoeffsApproach
  calculation of the expansion coefficients

• size_t numUncertainQuant
  number of invocations of quantify_uncertainty()

• int numSamplesOnModel
  number of truth samples performed on g_u_model to form the expansion

• int numSamplesOnExpansion
  expansion in order to estimate probabilities

**Private Attributes**

• Iterator expansionSampler
  an LHS sampling instance, but AIS could also be used.

• RealVector initialPtU
  stores the initial variables data in u-space
8.88.1 Detailed Description

colocation (SC)

The NonDExpansion class provides a base class for methods that use polynomial expansions to approximate the effect of parameter uncertainties on response functions of interest.

8.88.2 Member Function Documentation

8.88.2.1 void compute_statistics () [protected]

calculate analytic and numerical statistics from the expansion

Calculate analytic and numerical statistics from the expansion and log results within final_stats for use in OUU.

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

- NonDExpansion.H
- NonDExpansion.C
8.89  NonDGlobalEvidence Class Reference

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

Inheritance diagram for NonDGlobalEvidence::

```
  NonDGlobalEvidence
  |                  |
  v                  v
  NonDInterval
  |                  |
  v                  v
  NonDInterval
  |                  |
  v                  v
  NonDInterval
  |                  |
  v                  v
  NonDInterval
  |                  |
  v                  v
  NonDInterval
  |                  |
  v                  v
  NonDInterval
  |                  |
  v                  v
  NonDInterval
  |                  |
  v                  v
  NonD
  |                  |
  v                  v
  NonD
  |                  |
  v                  v
  NonD
  |                  |
  v                  v
  NonD
  |                  |
  v                  v
  NonD
  |                  |
  v                  v
  NonD
  |                  |
  v                  v
  NonD
  |                  |
  v                  v
  Iterator
  |                  |
  v                  v
  Analyzer
```

Public Member Functions

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

- **~NonDGlobalEvidence ()**  
  *destructor*

- **void initialize ()**  
  *perform any required initialization*

- **void set_cell_bounds ()**  
  *set the optimization variable bounds for each cell*

- **void get_best_sample (bool find_max, bool eval_approx)**  
  *determine truthFnStar and approxFnStar*

- **void post_process_cell_results (bool minimize)**  
  *post-process a cell minimization/maximization result*

- **void post_process_response_fn_results ()**  
  *post-process the interval computed for a response function*

- **void post_process_final_results ()**  
  *perform final post-processing*
8.89.1 Detailed Description

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

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

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

- NonDGlobalEvidence.H
- NonDGlobalEvidence.C
8.90 NonDGlobalInterval Class Reference

to calculate interval bounds for epistemic uncertainty quantification

Inheritance diagram for NonDGlobalInterval:

```
NonDGlobalInterval
  |       NonDInterval
  |        NonD
  |         NonDAnalyzer
  |          Iterator
  v
NonDGlobalEvidence
```

Public Member Functions

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

- **~NonDGlobalInterval ()**
  
  *destructor*

- **void quantify_uncertainty ()**
  
  *entire function or interval bounds on a particular statistical estimator*

Protected Member Functions

- **virtual void initialize ()**
  
  *perform any required initialization*

- **virtual void set_cell_bounds ()**
  
  *set the optimization variable bounds for each cell*

- **virtual void get_best_sample (bool find_max, bool eval_approx)**
  
  *determine truthFnStar and approxFnStar*

- **virtual void post_process_cell_results (bool minimize)**
post-process a cell minimization/maximization result

- virtual void post_process_response_fn_results ()
  post-process the interval computed for a response function

- virtual void post_process_final_results ()
  perform final post-processing

- void post_process_gp_results ()
  results, update convergence controls, and update GP approximation

Protected Attributes

- Iterator daceIterator
  LHS iterator for constructing initial GP for all response functions.

- Iterator gpOptimizer
  NCSU DIRECT optimizer for maximizing expected improvement.

- Model fHatModel
  GP model of response, one approximation per response function.

- Model eifModel
  max(EIF) sub-problem

- Real approxFnStar
  approximate response corresponding to minimum/maximum truth response

- Real truthFnStar
  minimum/maximum truth response function value

Static Private Member Functions

- static void EIF_objective_min (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  Expected Improvement Function (EIF) for minimizing the GP.

- static void EIF_objective_max (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  Expected Improvement Function (EIF) for maximizing the GP.
Private Attributes

- `const int seedSpec
  the user seed specification (default is 0)

- `int numSamples
  the number of samples used in the surrogate

- `String rngName
  name of the random number generator

- `size_t eifConvergenceCntr
  is less than the convergenceTol

- `size_t distConvergenceCntr
  in optimal solution is less than the convergenceTol

- `RealVector prevCStar
  stores previous optimal points for convergence

- `size_t sbIterNum
  surrogate-based minimization/maximization iteration count

- `bool approxConverged
  flag indicating convergence of a GP minimization or maximization

- `bool allResponsesPerIter
  flag for maximal response extraction

Static Private Attributes

- `static NonDGlobalInterval * nondGIInstance
  functions in order to avoid the need for static data

8.90.1 Detailed Description

to calculate interval bounds for epistemic uncertainty quantification

The NonDGlobalInterval class supports global nongradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an "inner loop" aleatory analysis such as intervals on means, variances, or percentile levels. The preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

The documentation for this class was generated from the following files:
• NonDGlobalInterval.H
• NonDGlobalInterval.C
8.91 NonDGlobalReliability Class Reference

Class for global reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDGlobalReliability::

```
Iterator
   
Analyzer
   
NonD
   
NonDReliability
   
NonDGlobalReliability
```

Public Member Functions

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

- **~NonDGlobalReliability ()**
  destructor

- **void quantify_uncertainty ()**
  approximations of the cumulative distribution function of response

- **void print_results (std::ostream &s)**
  MPP-search-based reliability methods.

Private Member Functions

- **void optimize_gaussian_process ()**
  construct the GP using EGO/SKO

- **void importance_sampling ()**
  perform multimodal adaptive importance sampling on the GP

- **void get_best_sample ()**
  improvement function in Performance Measure Approach (PMA)

- **Real constraint_penalty (const Real &constraint, const RealVector &c_variables)**
calculate the penalty to be applied to the PMA constraint value

- Real **expected\_improvement** (const RealVector &expected\_values, const RealVector &c\_variables)
  
  *expected improvement function for the GP*

- Real **expected\_feasibility** (const RealVector &expected\_values, const RealVector &c\_variables)
  
  *expected feasibility function for the GP*

### Static Private Member Functions

- static void **EIF\_objective\_eval** (const Variables &sub\_model\_vars, const Variables &recast\_vars, const Response &sub\_model\_response, Response &recast\_response)

  *Expected Improvement (EIF) problem formulation for PMA.*

- static void **EFF\_objective\_eval** (const Variables &sub\_model\_vars, const Variables &recast\_vars, const Response &sub\_model\_response, Response &recast\_response)

  *Expected Feasibility (EFF) problem formulation for RIA.*

### Private Attributes

- Real **fnStar**

  *minimum penalized response from among true function evaluations*

- short **meritFunctionType**

  *type of merit function used to penalize sample data*

- Real **lagrangeMult**

  *Lagrange multiplier for standard Lagrangian merit function.*

- Real **augLagrangeMult**

  *Lagrange multiplier for augmented Lagrangian merit function.*

- Real **penaltyParameter**

  *penalty parameter for augmented Lagrangian merit function*

- Real **lastConstraintViolation**

  *current iterate should be accepted (must reduce violation)*

- bool **lastIterateAccepted**

  *this controls update of parameters for augmented Lagrangian merit fn*
8.91 NonDGlobalReliability Class Reference

Static Private Attributes

- static NonDGlobalReliability * nondGlobRelInstance

functions in order to avoid the need for static data

8.91.1 Detailed Description

Class for global reliability methods within DAKOTA/UQ.

The NonDGlobalReliability class implements EGO/SKO for global MPP search, which maximizes an expected improvement function derived from Gaussian process models. Once the limit state has been characterized, a multimodal importance sampling approach is used to compute probabilities.

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

- NonDGlobalReliability.H
- NonDGlobalReliability.C
### 8.92 NonDGlobalSingleInterval Class Reference

to calculate interval bounds for epistemic uncertainty quantification

Inheritance diagram for NonDGlobalSingleInterval::

```
          Iterator
             |
             Analyzer
             |
             NonD
             |
      NonDInterval
             |
NonDGlobalInterval
             |
NonDGlobalSingleInterval
```

**Public Member Functions**

- `NonDGlobalSingleInterval (Model &model)`
  
  *constructor*

- `~NonDGlobalSingleInterval ()`

  *destructor*

**Protected Member Functions**

- void `initialize ()`
  
  *perform any required initialization*

- void `post_process_cell_results (bool minimize)`

  *post-process a cell minimization/maximization result*

- void `get_best_sample (bool find_max, bool eval_approx)`

  *determine truthFnStar and approxFnStar*

**Private Attributes**

- `size_t statCntr`
counter for finalStatistics

8.92.1 Detailed Description

to calculate interval bounds for epistemic uncertainty quantification

The NonDGlobalSingleInterval class supports global nongradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an "inner loop" aleatory analysis such as intervals on means, variances, or percentile levels. The preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

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

- NonDGlobalSingleInterval.H
- NonDGlobalSingleInterval.C
8.93 NonDIncremLHSSampling Class Reference

Performs incremental LHS sampling for uncertainty quantification.

Inheritance diagram for NonDIncremLHSSampling::

```
NonDIncremLHSSampling
    NonDSampling
        NonD
            Iterator
                Analyzer
                    NonD
                        NonDIncremLHSSampling
```

Public Member Functions

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

- **~NonDIncremLHSSampling ()**

  destructor

- **void quantify_uncertainty ()**

  parameter samples, and computing statistics on the ensemble of results.

- **void print_results (std::ostream &s)**

  print the final statistics

Static Protected Member Functions

- **static bool rank_sort (const int &x, const int &y)**

  sort algorithm to compute ranks for rank correlations

Private Attributes

- **int previousSamples**

  number of samples in previous LHS run
8.93 NonDIncremLHSSampling Class Reference

- bool varBasedDecompFlag
  
  flags computation of VBD

Static Private Attributes

- static RealArray rawData
  
  static data used by static rank_sort() fn

8.93.1 Detailed Description

Performs incremental LHS sampling for uncertainty quantification.

The Latin Hypercube Sampling (LHS) package from Sandia Albuquerque’s Risk and Reliability organization provides comprehensive capabilities for Monte Carlo and Latin Hypercube sampling within a broad array of user-specified probabilistic parameter distributions. The incremental LHS sampling capability allows one to supplement an initial sample of size n to size 2n while maintaining the correct stratification of the 2n samples and also maintaining the specified correlation structure. The incremental version of LHS will return a sample of size n, which when combined with the original sample of size n, allows one to double the size of the sample.

8.93.2 Constructor & Destructor Documentation

8.93.2.1 NonDIncremLHSSampling (Model & model)

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

8.93.3 Member Function Documentation

8.93.3.1 void quantify_uncertainty () [virtual]

parameter samples, and computing statistics on the ensemble of results.

Generate incremental samples. Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.

Implements NonD.

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

- NonDIncremLHSSampling.H
- NonDIncremLHSSampling.C
8.94 NonDIntegration Class Reference

umerical integration points for evaluation of expectation integrals

Inheritance diagram for NonDIntegration:

```
   NonDIntegration
   |                 |
   v                 v
  NonD              NonDIntegration
         |                |
         v                v
       NonDQuadrature  NonDSparseGrid
```

Public Member Functions

- virtual void initialize (const Pecos::ShortArray &u_types)
  initialize the integration approach

- const RealVector & weight_products () const
  return weightProducts

- const Real3DArray & gauss_points_array () const
  return gaussPts1D

- const Real3DArray & gauss_weights_array () const
  return gaussWts1D

- const Array< BasisPolynomial > & polynomial_basis () const
  return polynomialBasis

Protected Member Functions

- NonDIntegration (Model &model)
  constructor

- NonDIntegration (NoDBaseConstructor, Model &model)
  alternate constructor for instantiations "on the fly"

- ~NonDIntegration ()
8.94 NonDIntegration Class Reference

**Destructor**

- virtual void **check_variables** (const Pecos::ShortArray &x_types)
  verify self-consistency of variables data
- void **quantify_uncertainty** ()
  distributions into response statistics

**Protected Attributes**

- RealVector **weightProducts**
  n-dimensional stencil
- Real3DArray **gaussPts1D**
  numContinuousVars x num_levels_per_var sets of 1D Gauss points
- Real3DArray **gaussWts1D**
  numContinuousVars x num_levels_per_var sets of 1D Gauss weights
- Array< BasisPolynomial > **polynomialBasis**
  computing Gaussian quadrature points and weights

**Private Attributes**

- size_t **numIntegrations**
  counter for number of integration executions for this object

**8.94.1 Detailed Description**

Numerical integration points for evaluation of expectation integrals
This class provides a base class for shared code among NonDQuadrature and NonDSparseGrid.

**8.94.2 Constructor & Destructor Documentation**

**8.94.2.1 NonDIntegration (Model & model) [protected]**

Constructor
This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has
been called and probDescDB can be queried for settings from the method specification. It is not currently used,
as there are not yet separate nond_quadrature/nond_sparse_grid method specifications.
8.94.2.2 **NonDIntegration (NoDBBaseConstructor, Model & model)** [protected]

alternate constructor for instantiations "on the fly"
This alternate constructor is used for on-the-fly generation and evaluation of numerical integration points.

8.94.3 Member Function Documentation

8.94.3.1 **void initialize (const Pecos::ShortArray & u_types)** [virtual]
initialize the integration approach
Virtual function called from probDescDB-based constructors and from NonDIntegration::quantify_uncertainty()
Reimplemented in NonDQuadrature.

8.94.3.2 **void check_variables (const Pecos::ShortArray & x_types)** [protected, virtual]
verify self-consistency of variables data
Virtual function called from probDescDB-based constructors and from NonDIntegration::quantify_uncertainty()
The documentation for this class was generated from the following files:

- NonDIntegration.H
- NonDIntegration.C
8.95 NonDInterval Class Reference

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

Inheritance diagram for NonDInterval:

```
NonDInterval
  | NonD
  | Analyzer
  | NonDGlobalInterval NonDLHSInterval NonDLocalInterval
  | NonDGlobalEvidence NonDGlobalSingleInterval NonDLHSEvidence NonDLHSSingleInterval NonDLocalEvidence NonDLocalSingleInterval
```

Public Member Functions

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

- **~NonDInterval ()**
  
  *destructor*

- **void print_results (std::ostream &s)**
  
  *print the cumulative distribution functions for belief and plausibility*

Protected Member Functions

- **void initialize_final_statistics ()**
  
  *initialize finalStatistics for belief/plausibility results sets*

- **void compute_evidence_statistics ()**
  
  *or vice-versa*

- **void calculate_cells_and_bpas ()**
  
  *replaces CBPIIC_F77 from wrapper calculate_basic_prob_intervals()*

- **void calculate_cbf_cpf (bool complementary=true)**
  
  *plausibility replaces CCBFPF_F77 from wrapper calculate_cum_belief_plaus()*

Protected Attributes

- **bool singleIntervalFlag**
flag for SingleInterval derived class

- **RealVectorArray ccBelFn**
  Storage array to hold CCBF values.

- **RealVectorArray ccPlausFn**
  Storage array to hold CCPF values.

- **RealVectorArray ccBelVal**
  Storage array to hold CCB response values.

- **RealVectorArray ccPlausVal**
  Storage array to hold CCP response values.

- **RealVectorArray cellLowerBounds**
  Storage array to hold cell lower bounds.

- **RealVectorArray cellUpperBounds**
  Storage array to hold cell upper bounds.

- **Real2DArray cellFnLowerBounds**
  Storage array to hold cell min.

- **Real2DArray cellFnUpperBounds**
  Storage array to hold cell max.

- **RealArray cellBPA**
  Storage array to hold cell bpa.

- **size_t respFnCntr**
  response function counter

- **size_t cellCntr**
  cell counter

- **size_t numCells**
  total number of interval combinations

### 8.95.1 Detailed Description

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

The **NonDInterval** class implements the propagation of epistemic uncertainty using either pure interval propagation or Dempster-Shafer theory of evidence. In the latter approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated,
along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

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

- NonDInterval.H
- NonDInterval.C
8.96 NonDLHSEvidence Class Reference

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

Inheritance diagram for NonDLHSEvidence:

```plaintext
    Iterator
     |          
    Analyzer
     |          
        NonD
     |          
NonDLHSInterval
     |          
NonDInterval
     |    NonDLHSEvidence
```

Public Member Functions

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

- **~NonDLHSEvidence ()**
  
  *destructor*

- **void initialize ()**
  
  *perform any required initialization*

- **void post_process_samples ()**
  
  *post-process the output from executing lhsSampler*

8.96.1 Detailed Description

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

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.
The documentation for this class was generated from the following files:

- NonDLHSEvidence.H
- NonDLHSEvidence.C
8.97 NonDLHSInterval Class Reference

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

Inheritance diagram for NonDLHSInterval::

```
    Iterator
    |
  Analyzer
  |
  NonD
  |
NonDInterval
  |
NonDLHSInterval
  |
NonDLHSEvidence NonDLHSSingleInterval
```

**Public Member Functions**

- `NonDLHSInterval (Model &model)`
  
  *constructor*

- `~NonDLHSInterval ()`
  
  *destructor*

- `void quantify_uncertainty ()`
  
  *performs an epistemic uncertainty propagation using LHS samples*

**Protected Member Functions**

- `virtual void initialize ()`
  
  *perform any required initialization*

- `virtual void post_process_samples ()=0`
  
  *post-process the output from executing lhsSampler*

**Protected Attributes**

- `Iterator lhsSampler`
the LHS sampler instance

- const int seedSpec
  the user seed specification (default is 0)

- int numSamples
  the number of samples used

- String rngName
  name of the random number generator

### 8.97.1 Detailed Description

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

The `NonDLHSInterval` class implements the propagation of epistemic uncertainty using LHS-based methods.

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

- NonDLHSInterval.H
- NonDLHSInterval.C
8.98 NonDLHSSampling Class Reference

Performs LHS and Monte Carlo sampling for uncertainty quantification.

Inheritance diagram for NonDLHSSampling::

```
Public Member Functions

- NonDLHSSampling (Model &model)
  standard constructor

- NonDLHSSampling (Model &model, int samples, int seed, const String &rng, short sampling_vars_-mode=ACTIVE)
  alternate constructor for sample generation and evaluation "on the fly"

- NonDLHSSampling (int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation "on the fly"

- ~NonDLHSSampling ()
  destructor

Protected Member Functions

- void pre_run ()
  generate LHS samples in non-VBD cases

- void post_input ()
  read tabular data for post-run mode

- void quantify_uncertainty ()
  perform the evaluate parameter sets portion of run
```
8.98 NonDLHSSampling Class Reference

- void derived_post_run()
  generate statistics for LHS runs in non-VBD cases

- void print_results(std::ostream &s)
  print the final statistics

Private Attributes

- bool varBasedDecompFlag
  flags computation of VBD

8.98.1 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.

8.98.2 Constructor & Destructor Documentation

8.98.2.1 NonDLHSSampling (Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

8.98.2.2 NonDLHSSampling (Model & model, int samples, int seed, const String & rng, short sampling_vars_mode = ACTIVE)

alternate constructor for sample generation and evaluation "on the fly"

This alternate constructor is used for generation and evaluation of Model-based sample sets. A set_db_list_nodes has not been performed so required data must be passed through the constructor. It’s purpose is to avoid the need for a separate LHS specification within methods that use LHS sampling.

8.98.2.3 NonDLHSSampling (int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds)

alternate constructor for sample generation "on the fly"
This alternate constructor is used by `ConcurrentStrategy` for generation of uniform, uncorrelated sample sets. It is _not_ a letter-envelope instantiation and a set_db_list_nodes has not been performed. It is called with all needed data passed through the constructor and is designed to allow more flexibility in variables set definition (i.e., relax connection to a variables specification and allow sampling over parameter sets such as multiobjective weights). In this case, a `Model` is not used and the object must only be used for sample generation (no evaluation).

### 8.98.3 Member Function Documentation

#### 8.98.3.1 `void quantify_uncertainty()` [protected, virtual]

perform the evaluate parameter sets portion of run

Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.

Implements `NonD`.

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

- `NonDLHSSampling.H`
- `NonDLHSSampling.C`
8.99 NonDLHSSingleInterval Class Reference

Class for pure interval propagation using LHS.

Inheritance diagram for NonDLHSSingleInterval:

```
  NonDLHSSingleInterval
  |                     |
  v                     v
NonDLHSInterval
  |                     |
  v                     v
NonDInterval
  |                     |
  v                     v
NonD
  |                     |
  v                     v
Analyzer
  |                     |
  v                     v
Iterator
```

Public Member Functions

- NonDLHSSingleInterval (Model &model)
  
  constructor

- ~NonDLHSSingleInterval ()
  
  destructor

Protected Member Functions

- void initialize ()
  
  perform any required initialization

- void post_process_samples ()
  
  post-process the output from executing lhsSampler

Private Attributes

- size_t statCntr
  
  counter for finalStatistics
8.99.1 Detailed Description

Class for pure interval propagation using LHS.

The NonDSingleInterval class implements the propagation of epistemic uncertainty using ...

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

- NonDLHSSingleInterval.H
- NonDLHSSingleInterval.C
8.100 NonDLocalEvidence Class Reference

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

Inheritance diagram for NonDLocalEvidence:

```
  NonDLocalEvidence
  |______________
  NonDLocalInterval
  |______________
  NonDInterval
  |__________
  NonD
  |_____
  Analyzer
  |_____
  Iterator
```

Public Member Functions

- `NonDLocalEvidence (Model &model)`
  
  constructor

- `~NonDLocalEvidence ()`
  
  destructor

Protected Member Functions

- `void initialize ()`
  
  perform any required initialization

- `void set_cell_bounds ()`
  
  set the optimization variable bounds for each cell

- `void truncate_to_cell_bounds (RealVector &initial_pt)`
  
  truncate initial_pt to respect current cell lower/upper bounds

- `void post_process_cell_results (bool minimize)`
  
  post-process a cell minimization/maximization result

- `void post_process_response_fn_results ()`
post-process the interval computed for a response function

- void post_process_final_results()
  perform final post-processing

8.100.1 Detailed Description

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

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

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

- NonDLocalEvidence.H
- NonDLocalEvidence.C
8.101 NonDLocalInterval Class Reference

calculate interval bounds for epistemic uncertainty quantification

Inheritance diagram for NonDLocalInterval:

```
NonDLocalInterval
   `-- NonDInterval
       `-- NonD
       |     `-- Iterator
       |         `-- NonDAnalyzer

NonDLocalEvidence
NonDLocalSingleInterval
```

Public Member Functions

- **NonDLocalInterval** (Model &model)
  
  Constructor

- **~NonDLocalInterval ()
  
  Destructor

- **void quantify_uncertainty ()
  
  entire function or interval bounds on a particular statistical estimator

- **String uses_method () const
  
  return name of active optimizer method

- **void method_recourse ()
  
  perform an MPP optimizer method switch due to a detected conflict

Protected Member Functions

- **virtual void initialize ()
  
  perform any required initialization

- **virtual void set_cell_bounds ()

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
set the optimization variable bounds for each cell

- virtual void `truncate_to_cell_bounds` (RealVector &initial_pt)
  truncate initial_pt to respect current cell lower/upper bounds

- virtual void `post_process_cell_results` (bool minimize)
  post-process a cell minimization/maximization result

- virtual void `post_process_response_fn_results`
  post-process the interval computed for a response function

- virtual void `post_process_final_results`
  perform final post-processing

Protected Attributes

- `Iterator minMaxOptimizer`
  local gradient-based optimizer

- `Model minMaxModel`
  recast model with sign flip for maximizing

Static Private Member Functions

- static void `objective_min` (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  the interval lower bound

- static void `objective_max` (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  the interval upper bound

Private Attributes

- bool `npsolFlag`
  selection (NPSOL SQP or OPT++ NIP)

Static Private Attributes

- static `NonDLocalInterval * nondLIInstance`
  functions in order to avoid the need for static data
calculate interval bounds for epistemic uncertainty quantification

The NonDLocalInterval class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an "inner loop" aleatory analysis such as intervals on means, variances, or percentile levels.

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

- NonDLocalInterval.H
- NonDLocalInterval.C
8.102 NonDLocalReliability Class Reference

Class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDLocalReliability:

```
    NonDLocalReliability
    |                     NonD
    |                     NonDReliability
    |                     NonDLocalReliability
    |                     Iterator
    |                     Analyzer
```

**Public Member Functions**

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

- **~NonDLocalReliability ()**

  *destructor*

- **void quantify_uncertainty ()**

  *approximations of the cumulative distribution function of response*

- **void print_results (std::ostream &s)**

  *MPP-search-based reliability methods.*

- **String uses_method () const**

  *return name of active MPP optimizer*

- **void method_recourse ()**

  *perform an MPP optimizer method switch due to a detected conflict*

**Private Member Functions**

- **void initial_taylor_series ()**

  *Taylor-series approximation.*

- **void mean_value ()**
computation of approximate statistics and importance factors

- void **mpp_search** ()
  employ a search for the most probable point (AMV, AMV+, FORM, SORM)

- void **initialize_class_data** ()
  convenience function for initializing class scope arrays

- void **initialize_level_data** ()
  data for each response function prior to level 0

- void **initialize_mpp_search_data** ()
  data for each z/p/beta level for each response function

- void **update_mpp_search_data** (const Variables &vars_star, const Response &resp_star)
  z/p/beta level for each response function

- void **update_level_data** ()
  statistics following MPP convergence

- void **update_pma_reliability_level** ()
  generalized reliabilities by inverting second-order integrations

- void **update_limit_state_surrogate** ()
  to the data fit embedded within uSpaceModel

- void **assign_mean_data** ()
  from ranVarMeansX/U, fnValsMeanX, fnGradsMeanX, and fnHessiansMeanX

- void **dg_ds_eval** (const RealVector &x_vars, const RealVector &fn_grad_x, RealVector &final_stat_grad)
  convenience function for evaluating dg/ds

- Real **probability** (const Real &beta, bool cdf_flag)
  second-order integration

- Real **reliability** (const Real &p, bool cdf_flag)
  second-order integration

- bool **reliability_residual** (const Real &p, const Real &beta, const RealVector &kappa, Real &res)
  corrections using Newton’s method (called by reliability(p))

- Real **reliability_residual_derivative** (const Real &p, const Real &beta, const RealVector &kappa)
  probability corrections using Newton’s method (called by reliability(p))

- void **principal_curvatures** ()
  Compute the kappaU vector of principal curvatures from fnHessU.
Private Attributes

- RealVector fnGradX
  evaluation

- RealVector fnGradU
  Jacobian $dx/du$.

- RealSymMatrix fnHessX
  evaluation

- RealSymMatrix fnHessU
  Jacobian $dx/du$.

- RealVector kappaU
  transformation of fnHessU

- RealVector fnValsMeanX
  response function values evaluated at mean $x$

- RealMatrix fnGradsMeanX
  response function gradients evaluated at mean $x$

- RealSymMatrixArray fnHessiansMeanX
  response function Hessians evaluated at mean $x$

- RealVector ranVarMeansU
  vector of means for all uncertain random variables in $u$-space

- RealVector initialPtU
  initial guess for MPP search in $u$-space

- RealVector mostProbPointX
  location of MPP in $x$-space

- RealVector mostProbPointU
  location of MPP in $u$-space

- RealVectorArray prevMPPULev0
  initialPtU within RBDO.

- RealMatrix prevFnGradDLev0
  for level 0. Used for warm-starting initialPtU within RBDO.

- RealMatrix prevFnGradULev0
  for level 0. Used for warm-starting initialPtU within RBDO.
8.102 NonDLocalReliability Class Reference

- RealVector prevICVars
  previous design vector. Used for warm-starting initialPtU within RBDO.

- ShortArray prevCumASVLev0
  for warm-starting initialPtU within RBDO.

- bool npsolFlag
  selection (NPSOL SQP or OPT++ NIP)

- bool warmStartFlag
  flag indicating the use of warm starts

- bool nipModeOverrideFlag
  flag indicating the use of move overrides within OPT++ NIP

- bool curvatureDataAvailable
  mostProbPointU) is available for computing principal curvatures

- short integrationOrder
  integration order (1 or 2) provided by integration specification

- short secondOrderIntType
  type of second-order integration: Breitung, Hohenbichler-Rackwitz, or Hong

- Real curvatureThresh
  cut-off value for 1/sqrt() term in second-order probability corrections.

- short taylorOrder
  derived from hessianType

- RealMatrix impFactor
  importance factors predicted by MV

- int npsolDerivLevel
  fn, 2 = analytic grads of constraints, 3 = analytic grads of both.

- unsigned short warningBits
  set of warnings accumulated during execution

8.102.1 Detailed Description

Class for the reliability methods within DAKOTA/UQ.

The NonDLocalReliability class implements the following reliability methods through the support of different limit state approximation and integration options: mean value (MVFOSS/MVSOSS), advanced mean value.
method (AMV, AMV^2) in x- or u-space, iterated advanced mean value method (AMV+, AMV^2+) in x- or u-space, two-point adaptive nonlinearity approximation (TANA) in x- or u-space, first order reliability method (FORM), and second order reliability method (SORM). All options except mean value employ an optimizer (currently NPSOL SQP or OPT++ NIP) to solve an equality-constrained optimization problem for the most probable point (MPP). The MPP search may be formulated as the reliability index approach (RIA) for mapping response levels to reliabilities/probabilities or as the performance measure approach (PMA) for performing the inverse mapping of reliability/probability levels to response levels.

8.102.2 Member Function Documentation

8.102.2.1 void initial_taylor_series () [private]
Taylor-series approximation.
An initial first- or second-order Taylor-series approximation is required for MV/AMV/AMV+/TANA or for the case where meanStats or stdDevStats (from MV) are required within finalStatistics for subIterator usage of Non-DLocalReliability.

8.102.2.2 void initialize_class_data () [private]
convenience function for initializing class scope arrays
Initialize class-scope arrays and perform other start-up activities, such as evaluating median limit state responses.

8.102.2.3 void initialize_level_data () [private]
data for each response function prior to level 0
For a particular response function prior to the first z/p/beta level, initialize/warm-start optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

8.102.2.4 void initialize_mpp_search_data () [private]
data for each z/p/beta level for each response function
For a particular response function at a particular z/p/beta level, warm-start or reset the optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

8.102.2.5 void update_mpp_search_data (const Variables & vars_star, const Response & resp_star)
[private]
z/p/beta level for each response function
Includes case-specific logic for updating MPP search data for the AMV/AMV+/TANA/NO_APPROX methods.
8.102.2.6  void update_level_data ()  [private]

statistics following MPP convergence
Updates computedRespLevels/computedProbLevels/computedRelLevels, finalStatistics, warm start, and graphics data.

8.102.2.7  void update_pma_reliability_level ()  [private, virtual]

generalized reliabilities by inverting second-order integrations
For PMA SORM with prescribed p-level or prescribed generalized beta-level, requestedCDFRelLevel must be updated. This virtual function redefinition is called from NonDReliability::PMA_constraint_eval().
Reimplemented from NonDReliability.

8.102.2.8  void dg_ds_eval (const RealVector & x_vars, const RealVector & fn_grad_x, RealVector & final_stat_grad)  [private]

convenience function for evaluating dg/ds
Computes dg/ds where s = design variables. Supports potentially overlapping cases of design variable augmentation and insertion.

8.102.2.9  Real probability (const Real & beta, bool cdf_flag)  [private]

second-order integration
Converts beta into a probability using either first-order (FORM) or second-order (SORM) integration. The SORM calculation first calculates the principal curvatures at the MPP (using the approach in Ch. 8 of Haldar & Mahadevan), and then applies correction formulations from the literature (Breitung, Hohenbichler-Rackwitz, or Hong).

8.102.2.10 Real reliability (const Real & p, bool cdf_flag)  [private]

second-order integration
Converts a probability into a reliability using the inverse of the first-order or second-order integrations implemented in NonDLocalReliability::probability().
The documentation for this class was generated from the following files:

- NonDLocalReliability.H
- NonDLocalReliability.C
8.103 NonDLocalSingleInterval Class Reference

calculate interval bounds for epistemic uncertainty quantification

Inheritance diagram for NonDLocalSingleInterval:

```
    Iterator
     |      
     v      
  Analyzer
       |      
       v      
  NonD
       |      
       v      
NonDInterval
       |      
       v      
NonDLocalInterval
       |      
       v      
NonDLocalSingleInterval
```

Public Member Functions

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

- **~NonDLocalSingleInterval ()**
  *destructor*

Protected Member Functions

- **void initialize ()**
  *perform any required initialization*

- **void post_process_cell_results (bool minimize)**
  *post-process a cell minimization/maximization result*

Private Attributes

- **size_t statCntr**
  *counter for finalStatistics*
8.103.1 Detailed Description

calculate interval bounds for epistemic uncertainty quantification

The NonDLocalSingleInterval class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an "inner loop" aleatory analysis such as intervals on means, variances, or percentile levels.

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

- NonDLocalSingleInterval.H
- NonDLocalSingleInterval.C
8.104 NonDPolynomialChaos Class Reference

quantification

Inheritance diagram for NonDPolynomialChaos::

```
  Iterator
    Analyzer
      NonD
        NonDExpansion
          NonDPolynomialChaos
```

Public Member Functions

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

- **~NonDPolynomialChaos ()**
  
  destructor

- **void quantify_uncertainty ()**

  perform a forward uncertainty propagation using PCE methods

- **void print_results (std::ostream &s)**

  print the final statistics and PCE coefficient array

- **void initialize_expansion ()**

  initialize random variable definitions and final stats arrays

Private Attributes

- **String expansionImportFile**

  filename for import of chaos coefficients

- **int expansionTerms**

  user specification of PCE terms

- **RealMatrix pceGradsMeanX**
evaluated at the means (used as uncertainty importance metrics)

8.104.1 Detailed Description

quantification

The NonDPolynomialChaos class uses a polynomial chaos expansion (PCE) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the OrthogPolyApproximation class to manage multiple types of orthogonal polynomials within a Wiener-Askey scheme to PCE. It supports PCE coefficient estimation via sampling, quadrature, point-collocation, and file import.

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

- NonDPolynomialChaos.H
- NonDPolynomialChaos.C
NonDQuadrature Class Reference

normals/uniforms/exponentials/betas/gammas.

Inheritance diagram for NonDQuadrature:

```
    Iterator
     |  
    ---|---
     |  
    Analyzer
     |  
     |  
    NonD
     |  
    NonDIntegration
     |  
    NonDQuadrature
```

### Public Member Functions

- `NonDQuadrature (Model &model, const UShortArray &order)`
- `const UShortArray & quadrature_order () const`

```
return quadOrder
```

### Protected Member Functions

- `NonDQuadrature (Model &model)`

```
constructor
```

- `~NonDQuadrature ()`

```
destructor
```

- `void get_parameter_sets (const Model &model)`

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

- `void initialize (const Pecos::ShortArray &u_types)`

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

### Private Member Functions

- `void check_integration ()`

```
verify self-consistency of integration specification
```
Private Attributes

- **UShortArray quadOrderSpec**
  
  *the user specification for the number of Gauss points per dimension*

- **UShortArray quadOrder**
  
  *external requirements communicated through sampling_reset()*

### 8.105.1 Detailed Description

normals/uniforms/exponentials/betas/gammas.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of moments. It employs Gauss-Hermite, Gauss-Legendre, Gauss-Laguerre, Gauss-Jacobi and generalized Gauss-Laguerre quadrature for use with normal, uniform, exponential, beta, and gamma density functions and integration bounds. The abscissas and weights for one-dimensional integration are extracted from the appropriate OrthogonalPolynomial class and are extended to n-dimensions using a tensor product approach.

### 8.105.2 Constructor & Destructor Documentation

#### 8.105.2.1 **NonDQuadrature (Model & model, const UShortArray & order)**

This alternate constructor is used for on-the-fly generation and evaluation of numerical quadrature points.

#### 8.105.2.2 **NonDQuadrature (Model & model) [protected]**

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate nond_quadrature method specification.

### 8.105.3 Member Function Documentation

#### 8.105.3.1 **void initialize (const Pecos::ShortArray & u_types) [protected, virtual]**

Called from probDescDB-based constructors and from NonDIntegration::quantify_uncertainty(). Reimplemented from NonDIntegration.

#### 8.105.3.2 **void sampling_reset (int min_samples, int rec_samples, bool all_data_flag, bool stats_flag) [inline, protected, virtual]**

used by DataFitSurfModel::build_global() to publish the minimum number of points needed from the quadrature routine in order to build a particular global approximation.
Reimplemented from `Iterator`.

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

- NonDQuadrature.H
- NonDQuadrature.C
8.106 NonDReliability Class Reference

Base class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDReliability:

```
NonDReliability
   |   |
   |   V nonD
   V   V
Analyzer
   |   |
   |   V nonD
   V   V
NonD
```

```
V   V
NonD
```

```
V   V
NonDReliability
```

```
V   V
NonDGlobalReliability
```

```
V   V
NonDLocalReliability
```

**Protected Member Functions**

- NonDReliability (Model &model)
  
  constructor

- ~NonDReliability ()
  
  destructor

- void initialize_graphics (bool graph_2d, bool tabular_data, const String &tabular_file)
  
  initialize graphics customized for reliability methods

- virtual void update_pma_reliability_level ()
  
  update requestedCDFRelLevel for use in PMA_constraint_eval()

**Static Protected Member Functions**

- static void RIA_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  
  (MPP) with the objective function of \((\text{norm } u)^2\).

- static void RIA_constraint_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  
  (MPP) with the constraint of \(G(u) = \text{response level}\).

- static void PMA_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
(MPP) with the objective function of \( G(u) \).

- static void `PMA_constraint_eval` (const `Variables` &sub_model_vars, const `Variables` &recast_vars, const `Response` &sub_model_response, `Response` &recast_response)
  (MPP) with the constraint of \( \|u\|^2 = \beta^2 \).

- static void `PMA2_set_mapping` (const `ActiveSet` &recast_set, `ActiveSet` &sub_model_set)
  beta-bar constraint target update is required for second-order PMA

Protected Attributes

- Model `uSpaceModel`
  recastings and data fits

- Model `mppModel`
  `RecastModel` which formulates the optimization subproblem: RIA, PMA, EGO.

- Iterator `mppOptimizer`
  `Iterator` which optimizes the `mppModel`.

- short `mppSearchType`
  x/u-space TANA, x/u-space EGO, or NO_APPROX

- Iterator `importanceSampler`
  `Iterator` importance sampling instance used to compute/refine probabilities

- short `integrationRefinement`
  refinement specification

- size_t `numRelAnalyses`
  number of invocations of `quantify_uncertainty()`

- size_t `approxIters`
  number of approximation cycles for the current `respFnCount/levelCount`

- bool `approxConverged`
  indicates convergence of approximation-based iterations

- int `respFnCount`
  counter for which response function is being analyzed

- size_t `levelCount`
  counter for which response/probability level is being analyzed

- size_t `statCount`
counter for which final statistic is being computed

- Real requestedRespLevel
  the response level target for the current response function

- Real requestedCDFProbLevel
  the CDF probability level target for the current response function

- Real requestedCDFRelLevel
  the CDF reliability level target for the current response function

- Real computedRespLevel
  output response level calculated

- Real computedRelLevel
  output reliability level calculated

Static Protected Attributes

- static NonDReliability * nondRelInstance
  functions in order to avoid the need for static data

8.106.1 Detailed Description

Base class for the reliability methods within DAKOTA/UQ.

The NonDReliability class provides a base class for NonDLocalReliability, which implements traditional MPP-based reliability methods, and NonDGlobalReliability, which implements global limit state search using Gaussian process models in combination with multimodal importance sampling.

8.106.2 Member Function Documentation

8.106.2.1 void RIA_objective_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(MPP) with the objective function of \((\text{norm } u)^2\).

This function recasts a G(u) response set (already transformed and approximated in other recursions) into an RIA objective function.
8.106.2.2 void RIA_constraint_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(MPP) with the constraint of G(u) = response level.
This function recasts a G(u) response set (already transformed and approximated in other recursions) into an RIA equality constraint.

8.106.2.3 void PMA_objective_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(MPP) with the objective function of G(u).
This function recasts a G(u) response set (already transformed and approximated in other recursions) into an PMA objective function.

8.106.2.4 void PMA_constraint_eval (const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) [static, protected]

(MPP) with the constraint of (norm u)^2 = beta^2.
This function recasts a G(u) response set (already transformed and approximated in other recursions) into a PMA equality constraint.

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

- NonDReliability.H
- NonDReliability.C
8.107 NonDSampling Class Reference

NonDIncremLHSSampling, and NonDAdaptImpSampling.

Inheritance diagram for NonDSampling:

```
NonDSampling
  NonD
    Analyzer
      Iterator
        NonDAdaptImpSampling
        NonDIncremLHSSampling
        NonDLHSSampling
```

Public Member Functions

- void compute_distribution_mappings (const ResponseArray &samples)
  
  z to p/beta and of p/beta to z

- void update_final_statistics ()
  
  and computedProbLevels/computedRelLevels/computedRespLevels

- void print_distribution_mappings (std::ostream &s) const
  
  prints the p/beta/z mappings computed in compute_distribution_mappings()

Protected Member Functions

- NonDSampling (Model &model)
  
  constructor

- NonDSampling (NoDBBaseConstructor, Model &model, int samples, int seed, const String &rng)
  
  alternate constructor for sample generation and evaluation "on the fly"

- NonDSampling (NoDBBaseConstructor, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  alternate constructor for sample generation "on the fly"

- ~NonDSampling ()
  
  destructor
• void `sampling_reset` (int min_samples, int rec_samples, bool all_data_flag, bool stats_flag)
  resets number of samples and sampling flags

• const `String` & `sampling_scheme` () const
  return sampleType: "lhs" or "random"

• void `vary_pattern` (bool pattern_flag)
  set varyPattern

• void `get_parameter_sets` (const `Model` &model)
  distributions/bounds defined in the incoming model.

• void `get_parameter_sets` (const RealVector &lower_bnds, const RealVector &upper_bnds)
  lower_bnds/upper_bnds.

• void `initialize_lhs` (bool write_message)
  increments numLHSRuns, sets random seed, and initializes lhsDriver

• void `finalize_lhs` (RealMatrix &samples_array)
  converts samples_array into allVariables

• void `compute_statistics` (const VariablesArray &vars_samples, const ResponseArray &resp_samples)
  or intervals (epistemic or mixed uncertainties)

• void `compute_intervals` (const ResponseArray &samples)
  called by `compute_statistics()` to calculate min/max intervals

• void `compute_moments` (const ResponseArray &samples)
  deviations, and confidence intervals

• void `print_statistics` (std::ostream &s) const
  prints the statistics computed in `compute_statistics()`

• void `print_intervals` (std::ostream &s) const
  prints the intervals computed in `compute_intervals()`

• void `print_moments` (std::ostream &s) const
  prints the moments computed in `compute_moments()`

Protected Attributes

• const int `seedSpec`
  the user seed specification (default is 0)

• int `randomSeed`
the current seed

- `const int samplesSpec`
  initial specification of number of samples

- `int numSamples`
  the current number of samples to evaluate

- `String rngName`
  name of the random number generator

- `String sampleType`
  the sample type: random, lhs, or incremental_lhs

- `Pecos::LHSDriver lhsDriver`
  the C++ wrapper for the F90 LHS library

- `bool statsFlag`
  flags computation/output of statistics

- `bool allDataFlag`
  flags update of allVariables/allResponses

- `short samplingVarsMode`
  the sampling mode: ACTIVE, ACTIVE_UNIFORM, ALL, or ALL_UNIFORM

- `short sampleRanksMode`
  SET_RANKS, or SET_GET_RANKS.

- `bool varyPattern`
  repeatable

- `RealMatrix sampleRanks`
  data structure to hold the sample ranks

- `RealVector mean95CI Deltas`
  intervals (calculated in `compute_moments()`)
Private Attributes

- size_t numLHSRuns
  - counter for number of executions of get_parameter_sets() for this object

- RealVector minValues
  - (calculated in compute_intervals())

- RealVector maxValues
  - (calculated in compute_intervals())

8.107.1 Detailed Description

NonDIncremLHSSampling, and NonDAdaptImpSampling.

This base class provides common code for sampling methods which employ the Latin Hypercube Sampling (LHS) package from Sandia Albuquerque’s Risk and Reliability organization. NonDSampling now exclusively utilizes the 1998 Fortran 90 LHS version as documented in SAND98-0210, which was converted to a UNIX link library in 2001. The 1970’s vintage LHS (that had been f2c’d and converted to incomplete classes) has been removed.

8.107.2 Constructor & Destructor Documentation

8.107.2.1 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.

8.107.2.2 NonDSampling (NoDBBaseConstructor, Model & model, int samples, int seed, const String & rng) [protected]

alternate constructor for sample generation and evaluation "on the fly"

This alternate constructor is used for generation and evaluation of on-the-fly sample sets.

8.107.2.3 NonDSampling (NoDBBaseConstructor, int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds) [protected]

alternate constructor for sample generation "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets.
8.107.3 Member Function Documentation

8.107.3.1 void sampling_reset (int min_samples, int rec_samples, bool all_data_flag, bool stats_flag)
   [inline, protected, virtual]

resets number of samples and sampling flags
used by DataFitSurrModel::build_global() to publish the minimum number of samples needed from the sampling
routine (to build a particular global approximation) and to set allDataFlag and statsFlag. In this case, allData-
Flag is set to true (vectors of variable and response sets must be returned to build the global approximation) and
statsFlag is set to false (statistics computations are not needed).
Reimplemented from Iterator.

8.107.3.2 void get_parameter_sets (const Model & model) [protected, virtual]

distributions/bounds defined in the incoming model.
This version of get_parameter_sets() extracts data from the user-defined model in any of the four sampling modes.
Reimplemented from Analyzer.

8.107.3.3 void get_parameter_sets (const RealVector & lower_bnds, const RealVector & upper_bnds)
   [protected]

lower_bnds/upper_bnds.
This version of get_parameter_sets() does not extract data from the user-defined model, but instead relies on
the incoming bounded region definition. It only support a UNIFORM sampling mode, where the distinction of
ACTIVE_UNIFORM vs. ALL_UNIFORM is handled elsewhere.
The documentation for this class was generated from the following files:

- NonDSampling.H
- NonDSampling.C
8.108 NonDSparseGrid Class Reference

integrals over independent standard random variables.

Inheritance diagram for NonDSparseGrid::

```
NonDSparseGrid
   NonDIntegration
   NonD
   Analyzer
   Iterator
```

Public Member Functions

- **NonDSparseGrid** (Model &model, unsigned short level, const RealVector &dimension_pref)
- bool **isotropic_sparse_grid** () const
  
  \[ return \text{i}\text{s}otropiSSG \]

- unsigned short **sparse_grid_level** () const
  
  \[ return \text{ssgLevel} \]

- const RealVector & **sparse_grid_anisotropic_weights** () const
  
  \[ return \text{ssgAnisoLevelWts} \]

- const IntArray & **integration_rules** () const
  
  \[ return \text{integrationRules} \]

- const Real & **duplicate_tolerance** () const
  
  \[ return \text{duplicateTol} \]

- const IntArray & **unique_index_mapping** () const
  
  \[ return \text{uniqueIndexMapping} \]

- const Pecos::ShortArray & **integrated_variable_types** () const
  
  \[ return ProbabilityTransformation::\text{ranVarTypesU} \]

- void **level_to_order_closed_exponential** (unsigned short level, unsigned short &order) const
  
  \[ \text{with exponential growth} \]
• void \texttt{level\_to\_order\_open\_exponential} (unsigned short level, unsigned short &order) const
  \textit{with exponential growth}

• void \texttt{level\_to\_order\_open\_linear} (unsigned short level, unsigned short &order) const
  \textit{with linear growth}

• void \texttt{level\_to\_order} (const \texttt{UShortArray} &levels, \texttt{UShortArray} &orders) const
  \textit{integration orders based on integrationRules}

• void \texttt{initialize} (const Pecos::ShortArray &u_types, const \texttt{String} &sparse_grid_usage)

Protected Member Functions

• \texttt{NonDSparseGrid} (Model &model)
  \textit{constructor}

• \texttt{\textbar{}NonDSparseGrid} ()
  \textit{destructor}

• void \texttt{get\_parameter\_sets} (const Model &model)
  \textit{Returns one block of samples (ndim * num_samples)}.

• void \texttt{sampling\_reset} (int min\_samples, int rec\_samples, bool all\_data\_flag, bool stats\_flag)

Private Member Functions

• void \texttt{check\_integration} (const RealVector &dimension\_pref)
  \textit{verify self-consistency of integration specification}

Static Private Member Functions

• static void \texttt{bounded\_normal\_gauss\_points} (int order, int num\_params, double *params, double *data)
  \textit{BOUNDED\_NORMAL distribution}.

• static void \texttt{bounded\_normal\_gauss\_weights} (int order, int num\_params, double *params, double *data)
  \textit{BOUNDED\_NORMAL distribution}.

• static void \texttt{lognormal\_gauss\_points} (int order, int num\_params, double *params, double *data)
  \textit{function for numerically-generated Gauss points for LOGNORMAL distribution}

• static void \texttt{lognormal\_gauss\_weights} (int order, int num\_params, double *params, double *data)
  \textit{distribution}

• static void \texttt{bounded\_lognormal\_gauss\_points} (int order, int num\_params, double *params, double *data)
BOUNDLOGNORMAL distribution.

- static void bounded_lognormal_gauss_weights (int order, int num_params, double *params, double *data)

- static void loguniform_gauss_points (int order, int num_params, double *params, double *data)

- static void loguniform_gauss_weights (int order, int num_params, double *params, double *data)

- static void triangular_gauss_points (int order, int num_params, double *params, double *data)

- static void triangular_gauss_weights (int order, int num_params, double *params, double *data)

- static void gumbel_gauss_points (int order, int num_params, double *params, double *data)

- static void gumbel_gauss_weights (int order, int num_params, double *params, double *data)

- static void frechet_gauss_points (int order, int num_params, double *params, double *data)

- static void frechet_gauss_weights (int order, int num_params, double *params, double *data)

- static void weibull_gauss_points (int order, int num_params, double *params, double *data)

- static void weibull_gauss_weights (int order, int num_params, double *params, double *data)

- static void histogram_bin_gauss_points (int order, int num_params, double *params, double *data)

- static void histogram_bin_gauss_weights (int order, int num_params, double *params, double *data)
Private Attributes

- unsigned short ssgLevelSpec
  the user specification for the Smolyak sparse grid level

- unsigned short ssgLevel
  requirements communicated through sampling_reset()

- bool isotropicSSG
  flag indicating an isotropic Smolyak sparse grid

- RealVector ssgAnisoLevelWts
  to lower dimension preference due to lb <= |alpha|.i <= ub

- IntArray integrationRules
  integer codes for sgmga routine integration rule options

- IntArray numPolyParams
  (corresponds to set of variables defined by integrationRules)

- RealArray polyParams
  (corresponds to set of variables defined by integrationRules)

- Real duplicateTol
  duplication tolerance used in sgmga routines

- IntArray uniqueIndexMapping
  output from sgmga_unique_index()

- Array<FPType> compute1DPoints
  array of pointers to Gauss point evaluation functions

- Array<FPType> compute1DWeights
  array of pointers to Gauss weight evaluation functions

8.108.1 Detailed Description

integrals over independent standard random variables.

This class is used by NonDPolynomialChaos and NonDStochCollocation, but could also be used for general numerical integration of moments. It employs 1-D Clenshaw-Curtis and Gaussian quadrature rules within Smolyak sparse grids.
8.108.2 Constructor & Destructor Documentation

8.108.2.1 NonDSparseGrid (Model & model, unsigned short level, const RealVector & dimension_pref)

This alternate constructor is used for on-the-fly generation and evaluation of sparse grids within PCE and SC.

8.108.2.2 NonDSparseGrid (Model & model) [protected]

creator

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has
been called and probDescDB can be queried for settings from the method specification. It is not currently used,
as there is not a separate nond_sparse_grid method specification.

8.108.3 Member Function Documentation

8.108.3.1 void level_to_order_closed_exponential (unsigned short level, unsigned short & order) const

[inline]

with exponential growth
Adapted from webbur::level_to_order_default() for DAKOTA data types.

8.108.3.2 void level_to_order_open_exponential (unsigned short level, unsigned short & order) const

[inline]

with exponential growth
Adapted from webbur::level_to_order_default() for DAKOTA data types.

8.108.3.3 void level_to_order_open_linear (unsigned short level, unsigned short & order) const

[inline]

with linear growth
Adapted from webbur::level_to_order_default() for DAKOTA data types.

8.108.3.4 void initialize (const Pecos::ShortArray & u_types, const String & sparse_grid_usage)

Called from probDescDB-based constructors and from NonDIntegration::quantify_uncertainty()

8.108.3.5 void sampling_reset (int min_samples, int rec_samples, bool all_data_flag, bool stats_flag)

[protected, virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the sparse grid
routine in order to build a particular global approximation.
Reimplemented from Iterator.
8.108 NonDSparseGrid Class Reference

8.108.3.6 void check_integration (const RealVector & dimension_pref) [private]

verify self-consistency of integration specification
Called from probDescDB-based constructors and from NonDIntegration::quantify_uncertainty()

8.108.4 Member Data Documentation

8.108.4.1 bool isotropicSSG [private]

flag indicating an isotropic Smolyak sparse grid
sgmg routines are used for anisotropic, but sparse_grid_mixed_growth routines are used for isotropic due to reduced computational overhead

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

- NonDSparseGrid.H
- NonDSparseGrid.C
8.109 NonDStochCollocation Class Reference

The NonDStochCollocation class uses a stochastic collocation (SC) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the InterpPolyApproximation class to manage multidimensional Lagrange polynomial interpolants.

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

- NonDStochCollocation.H
- NonDStochCollocation.C
8.110 NPSOLOptimizer Class Reference

Wrapper class for the NPSOL optimization library.

Inheritance diagram for NPSOLOptimizer:

```
  Iterator
     |      
  Minimizer
     |      
  Optimizer   SOLBase
  NPSOLOptimizer
```

**Public Member Functions**

- **NPSOLOptimizer (Model &model)**
  
  *standard constructor*

- **NPSOLOptimizer (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor for Iterator instantiations by name*

- **NPSOLOptimizer (Model &model, const int &derivative_level, const Real &conv_tol)**
  
  *alternate constructor for instantiations "on the fly"*

  
  *alternate constructor for instantiations "on the fly"*

- **∼NPSOLOptimizer ()**
  
  *destructor*

- **void find_optimum ()**
  
  *Redefines the run virtual function for the optimizer branch.*

**Private Member Functions**

- **void find_optimum_on_model ()**
called by find_optimum for setUpType == “model”

- void find_optimum_on_user_functions()
  called by find_optimum for setUpType == “user_functions”

Static Private Member Functions

- static void objective_eval (int &mode, int &n, double *x, double &f, double *gradf, int &nstate)
  objective function (passed by function pointer to NPSOL).

Private Attributes

- String setUpType
  NonDReliability currently uses the user_functions mode.

- RealVector initialPoint
  holds initial point passed in for “user_functions” mode.

- RealVector lowerBounds
  holds variable lower bounds passed in for “user_functions” mode.

- RealVector upperBounds
  holds variable upper bounds passed in for “user_functions” mode.

- void(* userObjectiveEval)(int &, int &, double *, double &, double *, int &)
  “user_functions” mode.

- void(* userConstraintEval)(int &, int &, int &, int &, int *, int *, double *, double *, double *, int &)
  “user_functions” mode.

Static Private Attributes

- static NPSOLOptimizer * npsolInstance
  functions in order to avoid the need for static data

8.110.1 Detailed Description

Wrapper class for the NPSOL optimization library.

The NPSOLOptimizer class provides a wrapper for NPSOL, a Fortran 77 sequential quadratic programming library from Stanford University marketed by Stanford Business Associates. It uses a function pointer approach
for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function or accessed through a static pointer.

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

8.110.2 Constructor & Destructor Documentation

8.110.2.1 NPSOLOptimizer (Model & model)

standard constructor

This is the primary constructor. It accepts a Model reference.

8.110.2.2 NPSOLOptimizer (NoDBBaseConstructor, Model & model)

alternate constructor for Iterator instantiations by name

This is an alternate constructor which accepts a Model but does not have a supporting method specification from the ProblemDescDB.

8.110.2.3 NPSOLOptimizer (Model & model, const int & derivative_level, const Real & conv_tol)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

8.110.2.4 NPSOLOptimizer (const RealVector & initial_point, const RealVector & var_lower_bnds, const RealVector & var_upper_bnds, const RealMatrix & lin_ineq_coeffs, const RealVector & lin_ineq_lower_bnds, const RealVector & lin_ineq_upper_bnds, const RealMatrix & lin_eq_coeffs, const RealVector & lin_eq_targets, const RealVector & nonlin_ineq_lower_bnds, const RealVector & nonlin_ineq_upper_bnds, const RealVector & nonlin_eq_targets, void(*)(int &, int &, double *, double &, double *, int &) user_obj_eval, void(*)(int &, int &, int &, int &, int *, double *, double *, double *, int &) user_con_eval, const int & derivative_level, const Real & conv_tol)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers.

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

- NPSOLOptimizer.H
• NPSOL::Optimizer.C
8.111 NumericGenOrthogPolynomial Class Reference

orthogonal polynomials

Inheritance diagram for NumericGenOrthogPolynomial:

```
  BasisPolynomial
    |
    v
  OrthogonalPolynomial
    |
    v
  NumericGenOrthogPolynomial
```

**Public Member Functions**

- **NumericGenOrthogPolynomial ()**
  - default constructor

- **~NumericGenOrthogPolynomial ()**
  - destructor

- **const RealArray & gauss_points (unsigned short order)**
  - return the Gauss quadrature points corresponding to polynomial order

- **const RealArray & gauss_weights (unsigned short order)**
  - return the Gauss quadrature weights corresponding to polynomial order

- **void bounded_normal_distribution (const Real &mean, const Real &std_dev, const Real &l_bnd, const Real &u_bnd)**
  - set distribution type and parameters for a BOUNDED_NORMAL distribution

- **void lognormal_distribution (const Real &mean, const Real &std_dev)**
  - set distribution type and parameters for a LOGNORMAL distribution

- **void bounded_lognormal_distribution (const Real &mean, const Real &std_dev, const Real &l_bnd, const Real &u_bnd)**
  - set distribution type and parameters for a BOUNDED_LOGNORMAL distribution

- **void loguniform_distribution (const Real &l_bnd, const Real &u_bnd)**
  - set distribution type and parameters for a LOGUNIFORM distribution

- **void triangular_distribution (const Real &mode, const Real &l_bnd, const Real &u_bnd)**
  - set distribution type and parameters for a TRIANGULAR distribution

- **void gumbel_distribution (const Real &alpha, const Real &beta)**

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
set distribution type and parameters for a GUMBEL distribution

- void frechet_distribution (const Real &alpha, const Real &beta)
  set distribution type and parameters for a FRECHET distribution

- void weibull_distribution (const Real &alpha, const Real &beta)
  set distribution type and parameters for a WEIBULL distribution

- void histogram_bin_distribution (const RealVector &bin_pairs)
  set distribution type and parameters for a WEIBULL distribution

- void coefficients_norms_flag (bool flag)
  set coeffsNormsFlag

Protected Member Functions

- const Real & get_value (const Real &x, unsigned short order)
  for a given parameter x

- const Real & get_gradient (const Real &x, unsigned short order)
  for a given parameter x

- const Real & norm_squared (unsigned short order)
  return the inner product \(<NG_i,NG_i> = ||NG_i||^2\)

Private Member Functions

- void solve_eigenproblem (unsigned short m)
  points and weights for an orthogonal polynomial of order m

- void polynomial_recursion (RealVector &poly_coeffs_ip1, const Real &alpha_i, const RealVector &poly_coeffs_i, const Real &beta_i, const RealVector &poly_coeffs_im1)
  compute three point recursion for polyCoeffs[i+1]

- void polynomial_recursion (RealVector &poly_coeffs_ip1, const Real &alpha_i, const RealVector &poly_coeffs_i)
  compute truncated three point recursion for polyCoeffs[i+1]

- Real inner_product (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2)
  compute inner product of specified polynomial orders

- Real hermite_unbounded_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2, NGFPType weight_fn)
compute an unbounded integral using Gauss-Hermite integration

- Real fejer_unbounded_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2, NGFPType weight_fn)  
  change of variables

- Real laguerre_semibounded_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2, NGFPType weight_fn)  
  compute a semibounded integral using Gauss-Laguerre integration

- Real fejer_semibounded_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2, NGFPType weight_fn)  
  change of variables

- Real legendre_bounded_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2, NGFPType weight_fn, Real start, Real end)  
  Gauss-Legendre integration.

- Real cc_bounded_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2, NGFPType weight_fn, Real start, Real end)  
  Clenshaw-Curtis integration.

- Real riemann_bounded_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2, NGFPType weight_fn, Real start, Real end)  
  compute a bounded integral over the specified range using Riemann sums

- Real native_quadrature_integral (const RealVector &poly_coeffs1, const RealVector &poly_coeffs2)  
  (up to order 2m-1 based on gaussPoints and gaussWeights of order m)

- const Real & get_value (const Real &x, const RealVector &poly_coeffs)  
  coefficients for a given parameter value

- const Real & get_gradient (const Real &x, const RealVector &poly_coeffs)  
  coefficients with respect to its dimension for a given parameter value

Static Private Member Functions

- static Real bounded_normal_pdf (const Real &x, const RealVector &params)  
  thin wrapper for Pecos::bounded_normal_pdf for NGFPType API

- static Real lognormal_pdf (const Real &x, const RealVector &params)  
  thin wrapper for Pecos::lognormal_pdf for NGFPType API

- static Real bounded_lognormal_pdf (const Real &x, const RealVector &params)  
  thin wrapper for Pecos::bounded_lognormal_pdf for NGFPType API
• static Real \texttt{loguniform_pdf} (const Real &x, const RealVector &params)
  \textit{thin wrapper for Pecos::loguniform_pdf for NGFPType API}

• static Real \texttt{triangular_pdf} (const Real &x, const RealVector &params)
  \textit{thin wrapper for Pecos::triangular_pdf for NGFPType API}

• static Real \texttt{gumbel_pdf} (const Real &x, const RealVector &params)
  \textit{thin wrapper for Pecos::gumbel_pdf for NGFPType API}

• static Real \texttt{frechet_pdf} (const Real &x, const RealVector &params)
  \textit{thin wrapper for Pecos::frechet_pdf for NGFPType API}

• static Real \texttt{weibull_pdf} (const Real &x, const RealVector &params)
  \textit{thin wrapper for Pecos::weibull_pdf for NGFPType API}

**Private Attributes**

• short \texttt{distributionType}
  \textit{BOUNDED_LOGNORMAL, LOGUNIFORM, TRIANGULAR, GUMBEL, FRECHET, or WEIBULL.}

• RealVector \texttt{distParams}
  \textit{distribution parameters (e.g., mean, std\_dev, alpha, beta)}

• bool \texttt{coeffsNormsFlag}
  \textit{(if false, only gaussPoints and gaussWeights are computed)}

• RealVectorArray \texttt{polyCoeffs}
  \textit{coefficients of the orthogonal polynomials, from order 0 to m}

• RealVector \texttt{orthogPolyNormsSq}
  \textit{as defined by the inner product <Poly\_i, Poly\_i> = ||Poly\_i||^2}

### 8.111.1 Detailed Description

orthogonal polynomials

The \texttt{NumericGenOrthogPolynomial} class numerically generates a univariate orthogonal polynomial of a particular order, along with its Gauss points, Gauss weights, and norms. It uses a variety of algorithms due to Chebyshev and Stieltjes as reported by Golub and Welsch (Mathematics of Computation, Vol. 23, No. 106, 1969) and Gautschi (SIAM J. Sci. Stat. Comput., Vol. 3, No. 3, 1982). It enables (mixed) multidimensional orthogonal polynomial basis functions within \texttt{OrthogPolyApproximation}. 
8.111.2 Member Function Documentation

8.111.2.1 void solve_eigenproblem (unsigned short m) [private]

points and weights for an orthogonal polynomial of order m
Numbering conventions follow Gautschi.
The documentation for this class was generated from the following files:

- NumericGenOrthogPolynomial.H
- NumericGenOrthogPolynomial.C
8.112 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy.

Inheritance diagram for Optimizer::

```
      iterator
       /  \
     /    \
  Minimizer  \
       /     \
      /      \
    Optimizer

      APPSOptimizer
      /       \
    COLINOptimizer
    /        \
CONMINOptimizer
    /          \
DOTOptimizer
       /        \
    JEGAOptimizer
       /        \
    NCSUOptimizer
       /        \
NLPQLPOptimizer
       /        \
    NPSOLOptimizer
       /        \
    SNLLOptimizer
```

Protected Member Functions

- **Optimizer ()**
  
  *default constructor*

- **Optimizer (Model &model)**
  
  *standard constructor*

- **Optimizer (NoDBBaseConstructor, Model &model)**
  
  *alternate constructor for "on the fly" instantiations*
• **Optimizer** (NoDBBaseConstructor, size_t num_cv, size_t num_div, size_t num_drv, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq)
  alternate constructor for "on the fly" instantiations

• **~Optimizer** ()
  destructor

• void **derived_initialize_run** ()
• void **run** ()
  and may contain pre/post steps in lieu of separate pre/post

• void **derived_post_run** ()
• void **derived_finalize_run** ()
  portions of finalize_run specific to derived iterators

• void **print_results** (std::ostream &s)
• virtual void **find_optimum** ()=0
  Redefines the run virtual function for the optimizer branch.

### Protected Attributes

• size_t **numObjectiveFns**
  number of objective functions (iterator view)

• size_t **numUserObjectiveFns**
  number of objective functions (user's model view)

• bool **multiObjFlag**
  flag indicating whether multi-objective transformations are necessary

• Optimizer ***prevOptInstance**
  pointer containing previous value of optimizerInstance

### Static Protected Attributes

• static Optimizer ***optimizerInstance**
  pointer to Optimizer instance used in static member functions
Private Member Functions

- void weighted_sum (const Response &full_response, Response &reduced_response, const RealVector &wts) const
  
  weighted objective for single-objective optimizers

- void multi_objective_retrieve (const Variables &vars, Response &response) const
  
  from the solution of a single-objective optimizer

Static Private Member Functions

- static void primary_resp_recast (const Variables &native_vars, const Variables &scaled_vars, const Response &native_response, Response &scaled_response)
  
  from native (user) to iterator space

8.112.1 Detailed Description

Base class for the optimizer branch of the iterator hierarchy.

The Optimizer class provides common data and functionality for DOTOptimizer, CONMINOptimizer, NPSOLOptimizer, SNLLOptimizer, NLPLPOptimizer, COLINOptimizer, and JEGAOptimizer.

8.112.2 Constructor & Destructor Documentation

8.112.2.1 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.

8.112.3 Member Function Documentation

8.112.3.1 void derived_initialize_run () [protected, virtual]

Implements portions of initialize_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of derived_initialize_run() (which would otherwise hide it).

Reimplemented from Minimizer.

Reimplemented in CONMINOptimizer, DOTOptimizer, NLPLPOptimizer, and SNLLOptimizer.

8.112.3.2 void run () [inline, protected, virtual]

and may contain pre/post steps in lieu of separate pre/post
Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

8.112.3.3 void derived_post_run () [protected, virtual]

Implements portions of post_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of derived_post_run() (which would otherwise hide it).

Reimplemented from Iterator.
Reimplemented in SNLLOptimizer.

8.112.3.4 void derived_finalize_run () [inline, protected, virtual]

portions of finalize_run specific to derived iterators

Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of finalize_run(). Redefinition by derived classes is optional.

Reimplemented from Minimizer.
Reimplemented in SNLLOptimizer.

8.112.3.5 void print_results (std::ostream & s) [protected, virtual]

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

Reimplemented from Iterator.

8.112.3.6 void primary_resp_recast (const Variables & native_vars, const Variables & scaled_vars, const Response & native_response, Response & iterator_response) [static, private]

from native (user) to iterator space

Objective function map from user/native space to iterator/scaled/combined space using a RecastModel. If resizing the response, copies the constraint (secondary) data from native_response too

8.112.3.7 void weighted_sum (const Response & full_response, Response & reduced_response, const RealVector & multiobj_wts) const [private]

weighted objective for single-objective optimizers

This function is responsible for the mapping of multiple objective functions into a single objective for publishing to single-objective optimizers. Used in DOTOptimizer, NPSOLOptimizer, SNLLOptimizer, and SGOPTApplication on every function evaluation. The simple weighting approach (using primaryRespFnWts) is the only technique supported currently. The weightings are used to scale function values, gradients, and Hessians as needed.
8.112.3.8  void multi_objective_retrieve (const Variables & vars, Response & response) const

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
8.113 OrthogonalPolynomial Class Reference

Base class for the orthogonal polynomial class hierarchy.

Inheritance diagram for OrthogonalPolynomial::

```
BasePolynomial
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| OrthogonalPolynomial
|                     |
| GenLaguerreOrthogPolynomial HermiteOrthogPolynomial JacobiOrthogPolynomial LaguerreOrthogPolynomial LegendreOrthogPolynomial NumericGenOrthogPolynomial
```

Public Member Functions

- ~OrthogonalPolynomial ()
  default constructor

- void reset_gauss ()
  destroy history of Gauss pts/wts due to change in alpha/beta stats

- void gauss_check (unsigned short order)
  perform unit testing on the Gauss points/weights

Protected Attributes

- Real orthogPolyNormSq
  \( <Poly_n, Poly_n> = ||Poly_n||^2 \) (returned by norm_squared())

- RealArray gaussPoints
  \( x \) parameter values for which Poly_n(x) = 0

- RealArray gaussWeights
  Gaussian weights for one-dimensional Gaussian quadrature.

8.113.1 Detailed Description

Base class for the orthogonal polynomial class hierarchy.

The OrthogonalPolynomial class is the base class for the univariate orthogonal polynomial class hierarchy in DAKOTA. One instance of an OrthogonalPolynomial is created for each variable within a multidimensional orthogonal polynomial basis function (a vector of OrthogonalPolynomials is contained in OrthogPolyApproximation, which may be mixed and matched in, e.g., the Wiener-Askey scheme for polynomial chaos).

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

- OrthogonalPolynomial.H
- OrthogonalPolynomial.C
8.114 OrthogPolyApproximation Class Reference

approximation).
Inheritance diagram for OrthogPolyApproximation::

```
Approximation

BasisPolyApproximation

OrthogPolyApproximation
```

Public Member Functions

- **OrthogPolyApproximation ()**
  default constructor

- **~OrthogPolyApproximation ()**
  destructor

- **void expansion_terms (const int &exp_terms)**
  set numExpansionTerms

- **const int & expansion_terms () const**
  get numExpansionTerms

- **void distributions (const Pecos::ShortArray &u_types, const Model &model)**
  invoke distribution_types() and, if needed, distribution_parameters()

- **bool distribution_types (const Pecos::ShortArray &u_types)**
  allocate polynomialBasis and basisTypes based on u_types

- **void distribution_basis ()**
  allocate polynomialBasis based on basisTypes

- **void distribution_parameters (const Pecos::ShortArray &u_types, const Model &model)**
  pass distribution parameters from model to polynomialBasis

- **const Array< BasisPolynomial > & polynomial_basis () const**
  get polynomialBasis

- **void polynomial_basis (const Array< BasisPolynomial > &poly_basis)**
  set polynomialBasis
void coefficients_norms_flag (bool flag)
         set NumericGenOrthogPolynomial::coeffsNormsFlag

void resolve_inputs ()
         (numExpansionTerms and approxOrder) based on user input

void allocate_arrays ()
         initialize polynomialBasis, multiIndex, et al.

t size_t sparse_grid_terms (unsigned short ssg_level, const RealVector &ssg_aniso_wts)
         with the provided (anisotropic) sparse grid level specification

Static Public Member Functions

static void distributions (const Pecos::ShortArray &u_types, const Model &model, Array<BasisPolynomial> &poly_basis, ShortArray &basis_types)
         invoke distribution_types() and, if needed, distribution_parameters()          

static bool distribution_types (const Pecos::ShortArray &u_types, ShortArray &basis_types)
         allocate poly_basis and basis_types based on u_types

static void distribution_basis (const ShortArray &basis_types, Array<BasisPolynomial> &poly_basis)
         allocate poly_basis based on basis_types

static void distribution_parameters (const Pecos::ShortArray &u_types, const Model &model, Array<BasisPolynomial> &poly_basis)
         pass distribution parameters from model to poly_basis

Protected Member Functions

int min_coefficients () const
         build the derived class approximation type in numVars dimensions

void find_coefficients ()
         orthogonal polynomials

void print_coefficients (std::ostream &s) const
         print the coefficients for the expansion

void compute_global_sensitivity ()
         Performs global sensitivity analysis using Sobol' Indices.
• const Real & get_value (const RealVector &x)
  retrieve the response PCE value for a given parameter vector

• const RealVector & get_gradient (const RealVector &x)
  and default DVV

• const RealVector & get_gradient (const RealVector &x, const UIntArray &dvv)
  and given DVV

• const Real & get_mean ()
  return the mean of the PCE, treating all variables as random

• const Real & get_mean (const RealVector &x)
  treating a subset of the variables as random

• RealVector get_mean_gradient ()
  vector, treating all variables as random

• const RealVector & get_mean_gradient (const RealVector &x, const UIntArray &dvv)
  and given DVV, treating a subset of the variables as random

• const Real & get_variance ()
  return the variance of the PCE, treating all variables as random

• const Real & get_variance (const RealVector &x)
  treating a subset of the variables as random

• const RealVector & get_variance_gradient ()
  vector, treating all variables as random

• const RealVector & get_variance_gradient (const RealVector &x, const UIntArray &dvv)
  vector and given DVV, treating a subset of the variables as random

• const Real & get_covariance (const RealVector &exp_coeffs_2)
  return the covariance of the PCE, treating all variables as random

• const Real & norm_squared (size_t expansion_index)
  treating all variables as random

• const Real & norm_squared_random (size_t expansion_index)
  treating a subset of the variables as random
Private Member Functions

- **void sparse_grid_multi_index** (unsigned short ssg_level, const RealVector &ssg_aniso_wts, UShort2DArray &multi_index)
  
  *initialize multiIndex using a sparse grid expansion*

- **void quadrature_order_to_integral_order** (const UShortArray &quad_order, UShortArray &int_order)
  
  *convert quadrature orders to integrand orders*

- **void integral_order_to_expansion_order** (const UShortArray &int_order, UShortArray &exp_order)
  
  *convert integrand orders to expansion orders*

- **void sparse_grid_level_to_expansion_order** (unsigned short ssg_level, UShortArray &exp_order)
  
  *convert sparse grid levels to expansion orders*

- **void append_unique** (const UShort2DArray &tp_multi_index, UShort2DArray &multi_index)
  
  *appear in multi_index*

- **void update_pareto** (const UShort2DArray &new_pareto, UShort2DArray &total_pareto)
  
  *update the total Pareto set with new Pareto-optimal polynomial indices*

- **bool assess_dominance** (const UShort2DArray &new_pareto, const UShort2DArray &total_pareto)
  
  *assess whether new_pareto is dominated by total_pareto*

- **void assess_dominance** (const UShortArray &new_order, const UShortArray &existing_order, bool &new_dominated, bool &existing_dominated)
  
  *against an incumbent polynomial index set*

- **Real multivariate_polynomial** (const RealVector &x, size_t term)
  
  *evaluated at a particular parameter set*

- **void integration** ()
  
  *(expCoeffsSolnApproach is QUADRATURE or SPARSE_GRID)*

- **void regression** ()
  
  *(expCoeffsSolnApproach is REGRESSION)*

- **void expectation** ()
  
  *(expCoeffsSolnApproach is SAMPLING)*

- **void gradient_check** ()
  
  *cross-validates alternate gradient expressions*
Private Attributes

- int numExpansionTerms
  number of terms in Polynomial Chaos expansion (length of chaosCoeffs)

- ShortArray basisTypes
  NUMERICALLY_GENERATED.

- Array<BasisPolynomial> polynomialBasis
  constructing the multivariate orthogonal/interpolation polynomials

- UShort2DArray multiIndex
  of the multivariate orthogonal polynomials

- Real multiPolyNormSq
  norm-squared of one of the multivariate polynomial basis functions

- short quadratureExpansion
  TENSOR_PRODUCT Expansion.

- short sparseGridExpansion
  HEURISTIC_TOTAL_ORDER, or TENSOR_PRODUCT_SUM expansion.

8.114.1 Detailed Description

The OrthogPolyApproximation class provides a global approximation based on orthogonal polynomials. It is used primarily for polynomial chaos expansions (for stochastic finite element approaches to uncertainty quantification).

8.114.2 Member Function Documentation

8.114.2.1 size_t sparse_grid_terms (unsigned short ssg_level, const RealVector & ssg_aniso_wts)

with the provided (anisotropic) sparse grid level specification
Return the number of terms in a sparse-grid expansion.

8.114.2.2 const Real & get_mean () [protected, virtual]

return the mean of the PCE, treating all variables as random
In this case, all expansion variables are random variables and the mean of the expansion is simply the first chaos coefficient.
Implements BasisPolyApproximation.
8.114.2.3 const Real & get_mean (const RealVector & x) [protected, virtual]

treating a subset of the variables as random

In this case, a subset of the expansion variables are random variables and the mean of the expansion involves evaluating the expectation over this subset.

Implements BasisPolyApproximation.

8.114.2.4 RealVector get_mean_gradient () [protected, virtual]

vector, treating all variables as random

In this function, all expansion variables are random variables and any design/state variables are omitted from the expansion. In this case, the derivative of the expectation is the expectation of the derivative. The mixed derivative case (some design variables are inserted and some are augmented) requires no special treatment.

Implements BasisPolyApproximation.

8.114.2.5 const RealVector & get_mean_gradient (const RealVector & x, const UInteger & dvv) [protected, virtual]

and given DVV, treating a subset of the variables as random

In this function, a subset of the expansion variables are random variables and any augmented design/state variables (i.e., not inserted as random variable distribution parameters) are included in the expansion. In this case, the mean of the expansion is the expectation over the random subset and the derivative of the mean is the derivative of the remaining expansion over the non-random subset. This function must handle the mixed case, where some design/state variables are augmented (and are part of the expansion: derivatives are evaluated as described above) and some are inserted (derivatives are obtained from expansionCoeffGrads).

Implements BasisPolyApproximation.

8.114.2.6 const Real & get_variance () [protected, virtual]

return the variance of the PCE, treating all variables as random

In this case, all expansion variables are random variables and the variance of the expansion is the sum over all but the first term of the coefficients squared times the polynomial norms squared.

Implements BasisPolyApproximation.

8.114.2.7 const Real & get_variance (const RealVector & x) [protected, virtual]

treating a subset of the variables as random

In this case, a subset of the expansion variables are random variables and the variance of the expansion involves summations over this subset.

Implements BasisPolyApproximation.
8.114.2.8 const RealVector & get_variance_gradient () [protected, virtual]

vector, treating all variables as random

In this function, all expansion variables are random variables and any design/state variables are omitted from the expansion. The mixed derivative case (some design variables are inserted and some are augmented) requires no special treatment.

Implements BasisPolyApproximation.

8.114.2.9 const RealVector & get_variance_gradient (const RealVector & x, const UIntArray & dvv) [protected, virtual]

vector and given DVV, treating a subset of the variables as random

In this function, a subset of the expansion variables are random variables and any augmented design/state variables (i.e., not inserted as random variable distribution parameters) are included in the expansion. This function must handle the mixed case, where some design/state variables are augmented (and are part of the expansion) and some are inserted (derivatives are obtained from expansionCoeffGrads).

Implements BasisPolyApproximation.

8.114.2.10 void integration () [private]

(expCoeffsSolnApproach is QUADRATURE or SPARSE_GRID)

The coefficients of the PCE for the response are calculated using a Galerkin projection of the response against each multivariate orthogonal polynomial basis fn using the inner product ratio $\langle f, \Psi \rangle / \langle \Psi^2 \rangle$, where inner product $\langle a, b \rangle$ is the n-dimensional integral of $a \ast b \ast$weighting over the support range of the n-dimensional (composite) weighting function. 1-D quadrature rules are defined for specific 1-D weighting functions and support ranges and approximate the integral of $f \ast$weighting as the Sum_i of w_i f_i. To extend this to n-dimensions, a tensor product quadrature rule or Smolyak sparse grid rule is applied using the product of 1-D weightings applied to the n-dimensional stencil of points. It is not necessary to approximate the integral for the denominator numerically, since this is available analytically.

8.114.2.11 void regression () [private]

(expCoeffsSolnApproach is REGRESSION)

In this case, regression is used in place of Galerkin projection. That is, instead of calculating the PCE coefficients using inner product ratios, linear least squares is used to estimate the PCE coefficients which best match a set of response samples. This approach is also known as stochastic response surfaces. The least squares estimation is performed using DGELSS (SVD) or DGGLSE (equality-constrained) from LAPACK, based on the presence of an anchorPoint.

8.114.2.12 void expectation () [private]

(expCoeffsSolnApproach is SAMPLING)
The coefficients of the PCE for the response are calculated using a Galerkin projection of the response against each multivariate orthogonal polynomial basis fn using the inner product ratio \( \langle f, \Psi_i \rangle / \langle \Psi_i^1 \rangle^2 \), where inner product \( \langle a, b \rangle \) is the n-dimensional integral of a*b*weighting over the support range of the n-dimensional (composite) weighting function. When interpreting the weighting function as a probability density function, \( \langle a, b \rangle = \text{expected value of } a \times b \), which can be evaluated by sampling from the probability density function and computing the mean statistic. It is not necessary to compute the mean statistic for the denominator, since this is available analytically.

### 8.114.2.13 void gradient_check () [private]

cross-validates alternate gradient expressions

This test works in combination with DEBUG settings in (Legendre,Laguerre,Jacobi,GenLaguerre)OrthogPolynomial::get_gradient().

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

- OrthogPolyApproximation.H
- OrthogPolyApproximation.C
8.115 ParallelConfiguration Class Reference

collectively identify a particular multilevel parallel configuration.

Public Member Functions

- **ParallelConfiguration ()**
  
  default constructor

- **ParallelConfiguration (const ParallelConfiguration &pl)**
  
  copy constructor

- **~ParallelConfiguration ()**
  
  destructor

- **ParallelConfiguration & operator= (const ParallelConfiguration &pl)**
  
  assignment operator

- **const ParallelLevel & w_parallel_level () const**
  
  return the ParallelLevel corresponding to wPLIter

- **const ParallelLevel & si_parallel_level () const**
  
  return the ParallelLevel corresponding to siPLIter

- **const ParallelLevel & ie_parallel_level () const**
  
  return the ParallelLevel corresponding to iePLIter

- **const ParallelLevel & ea_parallel_level () const**
  
  return the ParallelLevel corresponding to eaPLIter

Private Member Functions

- **void assign (const ParallelConfiguration &pl)**
  
  assign the attributes of the incoming pl to this object

Private Attributes

- **short numParallelLevels**
  
  number of parallel levels

- **ParLevLIter wPLIter**
  
  improves modularity by avoiding explicit usage of MPI_COMM_WORLD)
• ParLevLIter siPLIter
  
  (there may be more than one per parallel configuration instance)

• ParLevLIter iePLIter
  
  (there can only be one)

• ParLevLIter eaPLIter
  
  (there can only be one)

Friends

• class ParallelLibrary
  
  streamline implementation

8.115.1 Detailed Description

collectively identify a particular multilevel parallel configuration.

Rather than containing the multilevel parallel configuration directly, ParallelConfiguration instead provides a set of list iterators which point into a combined list of ParallelLevels. This approach allows different configurations to reuse ParallelLevels without copying them. A list of ParallelConfigurations is contained in ParallelLibrary (ParallelLibrary::parallelConfigurations).

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

• ParallelLibrary.H
### 8.116 ParallelDirectApplicInterface Class Reference

plug-ins using `assign_rep()`.

Inheritance diagram for ParallelDirectApplicInterface::

```
  Interface
     |     |
     v     v
ApplicationInterface
     |     |
     v     v
DirectApplicInterface
     |     |
     v     v
ParallelDirectApplicInterface
```

**Public Member Functions**

- **ParallelDirectApplicInterface** (const `Dakota::ProblemDescDB &problem_db`, const `MPI_Comm &analysis_comm`)
  
  *constructor*

- **~ParallelDirectApplicInterface** ()
  
  *destructor*

**Protected Member Functions**

- int **derived_map_ac** (const `Dakota::String &ac_name`)
  
  *execute an analysis code portion of a direct evaluation invocation*

### 8.116.1 Detailed Description

plug-ins using `assign_rep()`.

The plug-in `ParallelDirectApplicInterface` resides in namespace `SIM` and uses a copy of `textbook()` to perform parallel parameter to response mappings. It may be activated by specifying the `--with-plugin` configure option, which activates the `DAKOTA_PLUGIN` macro in `dakota_config.h` used by `main.C` (which activates the plug-in code block within that file) and activates the `PLUGIN_S` declaration defined in `Makefile.include` and used in `Makefile.source` (which add this class to the build). Test input files should then use an analysis_driver of "plugin_textbook".

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

- `PluginParallelDirectApplicInterface.H`
- `PluginParallelDirectApplicInterface.C`
8.117 ParallelLevel Class Reference

communicator partitioning.

Public Member Functions

- **ParallelLevel ()**
  - default constructor

- **ParallelLevel (const ParallelLevel &pl)**
  - copy constructor

- **~ParallelLevel ()**
  - destructor

- **ParallelLevel & operator= (const ParallelLevel &pl)**
  - assignment operator

- **bool dedicated_master_flag () const**
  - return dedicatedMasterFlag

- **bool communicator_split_flag () const**
  - return commSplitFlag

- **bool server_master_flag () const**
  - return serverMasterFlag

- **bool message_pass () const**
  - return messagePass

- **const int & num_servers () const**
  - return numServers

- **const int & processors_per_server () const**
  - return procsPerServer

- **const MPI_Comm & server_intra_communicator () const**
  - return serverIntraComm

- **const int & server_communicator_rank () const**
  - return serverCommRank

- **const int & server_communicator_size () const**
  - return serverCommSize
• const MPI_Comm & hub_server_intra_communicator () const
  return hubServerIntraComm

• const int & hub_server_communicator_rank () const
  return hubServerCommRank

• const int & hub_server_communicator_size () const
  return hubServerCommSize

• const MPI_Comm & hub_server_inter_communicator () const
  return hubServerInterComm

• MPI_Comm * hub_server_inter_communicators () const
  return hubServerInterComms

• const int & server_id () const
  return serverId

**Private Member Functions**

• void assign (const ParallelLevel &pl)
  assign the attributes of the incoming pl to this object

**Private Attributes**

• bool dedicatedMasterFlag
  signals dedicated master partitioning

• bool commSplitFlag
  signals a communicator split was used

• bool serverMasterFlag
  identifies master server processors

• bool messagePass
  flag for message passing at this level

• int numServers
  number of servers

• int procsPerServer
  processors per server
- MPI_Comm serverIntraComm
  intracomm. for each server partition

- int serverCommRank
  rank in serverIntraComm

- int serverCommSize
  size of serverIntraComm

- MPI_Comm hubServerIntraComm
  intracomm for all serverCommRank==0 w/i next higher level serverIntraComm

- int hubServerCommRank
  rank in hubServerIntraComm

- int hubServerCommSize
  size of hubServerIntraComm

- MPI_Comm hubServerInterComm
  intercomm. between a server & the hub (on server partitions only)

- MPI_Comm * hubServerInterComms
  intercomm. array on hub processor

- int serverId
  server identifier

Friends

- class ParallelLibrary
  streamline implementation

8.117.1 Detailed Description

communicator partitioning.

A list of these levels is contained in ParallelLibrary (ParallelLibrary::parallelLevels), which defines all of the parallelism levels across one or more multilevel parallelism configurations.

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

- ParallelLibrary.H
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)
• void **specify_outputs_restart** (const char *clh_std_output_filename=NULL, const char *clh_std_error_filename=NULL, const char *clh_read_restart_filename=NULL, const char *clh_write_restart_filename=NULL, int stop_restart_evals=0, bool pre_run_flag=false)
  
  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

• bool **command_line_check** () const
  
  return checkFlag

• bool **command_line_pre_run** () const
  
  return preRunFlag

• bool **command_line_run** () const
  
  return runFlag

• bool **command_line_post_run** () const
  
  return postRunFlag

• bool **command_line_user_modes** () const
  
  return userModesFlag

• const **String & command_line_pre_run_input** () const
  
  preRunInput filename

• const **String & command_line_pre_run_output** () const
  
  preRunOutput filename

• const **String & command_line_run_input** () const
  
  runInput filename

• const **String & command_line_run_output** () const
  
  runOutput filename

• const **String & command_line_post_run_input** () const
  
  postRunInput filename

• const **String & command_line_post_run_output** () const
  
  postRunOutput filename
- void send_si (int &send_int, int dest, int tag)
  
  blocking send at the strategy-iterator communication level

- void recv_si (int &recv_int, int source, int tag, MPI_Status &status)
  
  blocking receive at the strategy-iterator communication level

- void send_si (MPIPackBuffer &send_buff, int dest, int tag)
  
  blocking send at the strategy-iterator communication level

- void isend_si (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req)
  
  nonblocking send at the strategy-iterator communication level

- void recv_si (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status)
  
  blocking receive at the strategy-iterator communication level

- void irecv_si (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req)
  
  nonblocking receive at the strategy-iterator communication level

- void send_ie (MPIPackBuffer &send_buff, int dest, int tag)
  
  blocking send at the iterator-evaluation communication level

- void isend_ie (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req)
  
  nonblocking send at the iterator-evaluation communication level

- void recv_ie (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status)
  
  blocking receive at the iterator-evaluation communication level

- void irecv_ie (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req)
  
  nonblocking receive at the iterator-evaluation communication level

- void send_ea (int &send_int, int dest, int tag)
  
  blocking send at the evaluation-analysis communication level

- void isend_ea (int &send_int, int dest, int tag, MPI_Request &send_req)
  
  nonblocking send at the evaluation-analysis communication level

- void recv_ea (int &recv_int, int source, int tag, MPI_Status &status)
  
  blocking receive at the evaluation-analysis communication level

- void irecv_ea (int &recv_int, int source, int tag, MPI_Request &recv_req)
  
  nonblocking receive at the evaluation-analysis communication level

- void bcast_w (int &data)
  
  broadcast an integer across MPI_COMM_WORLD

- void bcast_j (int &data)
broadcast an integer across an iterator communicator

- void `bcast_i` (short &data)
  broadcast a short integer across an iterator communicator

- void `bcast_e` (int &data)
  broadcast an integer across an evaluation communicator

- void `bcast_a` (int &data)
  broadcast an integer across an analysis communicator

- void `bcast_si` (int &data)
  broadcast an integer across a strategy-iterator intra communicator

- void `bcast_w` (MPIPackBuffer &send_buff)
  broadcast a packed buffer across MPI_COMM_WORLD

- void `bcast_i` (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an iterator communicator

- void `bcast_e` (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an evaluation communicator

- void `bcast_a` (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an analysis communicator

- void `bcast_si` (MPIPackBuffer &send_buff)
  broadcast a packed buffer across a strategy-iterator intra communicator

- void `bcast_w` (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer broadcast across MPI_COMM_WORLD

- void `bcast_i` (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an iterator communicator

- void `bcast_e` (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an evaluation communicator

- void `bcast_a` (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an analysis communicator

- void `bcast_si` (MPIUnpackBuffer &recv_buff)
  matching recv for packed buffer bcast across a strat-iterator intra comm

- void `barrier_w` ()
  enforce MPI_Barrier on MPI_COMM_WORLD
• void `barrier_i()`
  enforce MPI_Barrier on an iterator communicator

• void `barrier_e()`
  enforce MPI_Barrier on an evaluation communicator

• void `barrier_a()`
  enforce MPI_Barrier on an analysis communicator

• void `reduce_sum_ea(double *local_vals, double *sum_vals, const int &num_vals)`
  compute a sum over an eval-analysis intra-communicator using MPI_Reduce

• void `reduce_sum_a(double *local_vals, double *sum_vals, const int &num_vals)`
  compute a sum over an analysis communicator using MPI_Reduce

• void `test(MPI_Request &request, int &test_flag, MPI_Status &status)`
  test a nonblocking send/receive request for completion

• void `wait(MPI_Request &request, MPI_Status &status)`
  wait for a nonblocking send/receive request to complete

• void `waitall(const int &num_recvs, MPI_Request *&recv_reqs)`
  wait for all messages from a series of nonblocking receives

• void `waitsome(const int &num_sends, MPI_Request *&recv_requests, int &num_recvs, int *&index_array, MPI_Status *&status_array)`
  but complete all that are available

• void `free(MPI_Request &request)`
  free an MPI_Request

• const int & `world_size()` const
  return worldSize

• const int & `world_rank()` const
  return worldRank

• bool `mpirun_flag()` const
  return mpirunFlag

• bool `is_null()` const
  return dummyFlag

• Real `parallel_time()` const
  returns current MPI wall clock time
• void parallel_configuration_iterator (const ParConfigLIter &pc_iter)
  set the current ParallelConfiguration node

• const ParConfigLIter & parallel_configuration_iterator () const
  return the current ParallelConfiguration node

• const ParallelConfiguration & parallel_configuration () const
  return the current ParallelConfiguration instance

• size_t num_parallel_configurations () const
  returns the number of entries in parallelConfigurations

• bool parallel_configuration_is_complete ()
  identifies if the current ParallelConfiguration has been fully populated

• void increment_parallel_configuration ()
  add a new node to parallelConfigurations and increment currPCIter

• bool w_parallel_level_defined () const
  parallel level

• bool si_parallel_level_defined () const
  strategy-iterator parallel level

• bool ie_parallel_level_defined () const
  iterator-evaluation parallel level

• bool ea_parallel_level_defined () const
  evaluation-analysis parallel level

• Array< MPI_Comm > analysis_intra_communicators ()
  prior to execution time.

Static Public Member Functions

• static bool detect_parallel_launch (int &argc, char **&argv)
  based on command line arguments and environment variables

Private Member Functions

• void init_communicators (const ParallelLevel &parent_pl, const int &num_servers, const int &procs_per_server, const int &max_concurrency, const int &asynch_local_concurrency, const String &default_config, const String &scheduling_override)
split a parent communicator into child server communicators

- void free_communicators (ParallelLevel &pl)
  deallocate intra/inter communicators for a particular ParallelLevel

- bool split_communicator_dedicated_master (const ParallelLevel &parent_pl, ParallelLevel &child_pl, const int &proc_remainder)
  and num_servers child communicators

- bool split_communicator_peer_partition (const ParallelLevel &parent_pl, ParallelLevel &child_pl, const int &proc_remainder)
  communicators (no dedicated master processor)

- bool resolve_inputs (int &num_servers, int &procs_per_server, const int &avail_procs, int &proc_remainder, const int &max_concurrency, const int &capacity_multiplier, const String &default_config, const String &scheduling_override, bool print_rank)
  resolve user inputs into a sensible partitioning scheme

- void send (MPIPackBuffer &send_buff, const int &dest, const int &tag, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking buffer send at the current communication level

- void send (int &send_int, const int &dest, const int &tag, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking integer send at the current communication level

- void isend (MPIPackBuffer &send_buff, const int &dest, const int &tag, MPI_Request &send_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking buffer send at the current communication level

- void isend (int &send_int, const int &dest, const int &tag, MPI_Request &send_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking integer send at the current communication level

- void recv (MPIUnpackBuffer &recv_buff, const int &source, const int &tag, MPI_Status &status, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking buffer receive at the current communication level

- void recv (int &recv_int, const int &source, const int &tag, MPI_Status &status, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  blocking integer receive at the current communication level

- void irecv (MPIUnpackBuffer &recv_buff, const int &source, const int &tag, MPI_Request &recv_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking buffer receive at the current communication level

- void irecv (int &recv_int, const int &source, const int &tag, MPI_Request &recv_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking integer receive at the current communication level
void irecv (int &recv_int, const int &source, const int &tag, MPI_Request &recv_req, ParallelLevel &parent_pl, ParallelLevel &child_pl)
  nonblocking integer receive at the current communication level

void bcast (int &data, const MPI_Comm &comm)
  broadcast an integer across a communicator

void bcast (short &data, const MPI_Comm &comm)
  broadcast a short integer across a communicator

void bcast (MPIPackBuffer &send_buff, const MPI_Comm &comm)
  send a packed buffer across a communicator using a broadcast

void bcast (MPIUnpackBuffer &recv_buff, const MPI_Comm &comm)
  matching receive for a packed buffer broadcast

void barrier (const MPI_Comm &comm)
  enforce MPI_Barrier on comm

void reduce_sum (double *local_vals, double *sum_vals, const int &num_vals, const MPI_Comm &comm)
  compute a sum over comm using MPI_Reduce

void check_error (const String &err_source, const int &err_code)
  check the MPI return code and abort if error

void manage_run_modes (CommandLineHandler &cmd_line_handler)
  manage run mode information from command-line handler

void split_filenames (const char *filenames, String &input_filename, String &output_filename)
  unchanged strings if tokens not found

Private Attributes

std::ofstream output_ofstream
  tagged file redirection of stdout

std::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

- bool `checkFlag`
  flags invocation with command line option `-check`

- bool `preRunFlag`
  flags invocation with command line option `-pre_run`

- bool `runFlag`
  flags invocation with command line option `-run`

- bool `postRunFlag`
  flags invocation with command line option `-post_run`

- bool `userModesFlag`
  whether user run mdoes are active

- String `preRunInput`
  filename for pre_run input

- String `preRunOutput`
  filename for pre_run output

- String `runInput`
  filename for run input

- String `runOutput`
  filename for run output

- String `postRunInput`
  filename for post_run input
- **String** `postRunOutput`
  filename for post_run output

- **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** `stopRestartEvals`
  number of evals at which to stop restart processing

- **List< ParallelLevel >** `parallelLevels`
  parallelism among one or more configurations

- **List< ParallelConfiguration >** `parallelConfigurations`
  indexing into `parallelLevels`

- **ParLevLIter** `currPLIter`
  list iterator identifying the current node in `parallelLevels`

- **ParConfigLIter** `currPConIter`
  list iterator identifying the current node in `parallelConfigurations`
8.118 ParallelLibrary Class Reference

8.118.1 Detailed Description

message passing within these levels.

The ParallelLibrary class encapsulates all of the details of performing message passing within multiple levels of parallelism. It provides functions for partitioning of levels according to user configuration input and functions for passing messages within and across MPI communicators for each of the parallelism levels. If support for other message-passing libraries beyond MPI becomes needed (PVM, ...), then ParallelLibrary would be promoted to a base class with virtual functions to encapsulate the library-specific syntax.

8.118.2 Constructor & Destructor Documentation

8.118.2.1 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.

8.118.2.2 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.

8.118.2.3 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.

8.118.3 Member Function Documentation

8.118.3.1 void specify_outputs_restart (CommandLineHandler & cmd_line_handler)

inputs (normal mode)
On the rank 0 processor, get the -output, -error, -read_restart, and -write_restart filenames and the -stop_restart limit from the command line. Defaults for the filenames from the command line handler are NULL for the filenames except write which defaults to dakota.rst and 0 for read_restart_evals if no user specification. This information is Bcast from rank 0 to all iterator masters in manage_outputs_restart().
8.118.3.2 void specify_outputs_restart (const char * clh_std_output_filename = NULL, const char * clh_std_error_filename = NULL, const char * clh_read_restart_filename = NULL, const char * clh_write_restart_filename = NULL, int stop_restart_evals = 0, bool pre_run_flag = false)

inputs (library mode).

Rather than extracting from the command line, pass the std output, std error, read restart, and write restart filenames and the stop restart limit directly. This function only needs to be invoked to specify non-default values [defaults for the filenames are NULL (resulting in no output redirection, no restart read, and default restart write) and 0 for the stop restart limit (resulting in no restart read limit)].

8.118.3.3 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.

8.118.3.4 void close_streams ()

close streams, files, and any other services

Close streams associated with manage_outputs and manage_restart and terminate any additional services that may be active.

8.118.3.5 void increment_parallel_configuration () [inline]

add a new node to parallelConfigurations and increment currPCIter

Called from the ParallelLibrary ctor and from Model::init_communicators(). An increment is performed for each Model initialization except the first (which inherits the world and strategy-iterator parallel levels from the first partial configuration).

8.118.3.6 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.
8.118.3.7 bool resolve_inputs (int & num_servers, int & procs_per_server, const int & avail_procs, int & proc_remainder, const int & max_concurrency, const int & capacity_multiplier, const String & default_config, const String & scheduling_override, bool print_rank) [private]

resolve user inputs into a sensible partitioning scheme

This function is responsible for the "auto-configure" intelligence of DAKOTA. It resolves a variety of inputs and overrides into a sensible partitioning configuration for a particular parallelism level. It also handles the general case in which a user's specification request does not divide out evenly with the number of available processors for the level. If num_servers & procs_per_server are both nondefault, then the former takes precedence.

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

- ParallelLibrary.H
- ParallelLibrary.C
8.119  ParamResponsePair Class Reference

evaluation id.

Public Member Functions

- **ParamResponsePair ()**
  
  *default constructor*

- **ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, bool deep_copy=false)**
  
  *alternate constructor for temporaries*

- **ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, const int eval_id, bool deep_copy=true)**
  
  *standard constructor for history uses*

- **ParamResponsePair (const ParamResponsePair &pair)**
  
  *copy constructor*

- **~ParamResponsePair ()**
  
  *destructor*

- **ParamResponsePair & operator= (const ParamResponsePair &pair)**
  
  *assignment operator*

- **void read (std::istream &s)**
  
  *read a ParamResponsePair object from an std::istream*

- **void write (std::ostream &s) const**
  
  *write a ParamResponsePair object to an std::ostream*

- **void read.annotated (std::istream &s)**
  
  *read a ParamResponsePair object in annotated format from an std::istream*

- **void write.annotated (std::ostream &s) const**
  
  *write a ParamResponsePair object in annotated format to an std::ostream*

- **void write.tabular (std::ostream &s) const**
  
  *write a ParamResponsePair object in tabular format to an std::ostream*

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

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

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

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

- int `eval_id` () const
  return the evaluation identifier

- const String & `interface_id` () const
  return the interface identifier from the response object

- const IntStringPair & `eval_interface_ids` () const
  return the aggregate eval/interface identifier from the response object

- const Variables & `prp_parameters` () const
  return the parameters object

- const Response & `prp_response` () const
  return the response object

- void `prp_response` (const Response &response)
  set the response object

- const ActiveSet & `active_set` () const
  return the active set object from the response object

- void `active_set` (const ActiveSet &set)
  set the active set object within the response object

**Private Attributes**

- Variables prPairParameters
  the set of parameters for the function evaluation

- Response prPairResponse
  the response set for the function evaluation

- IntStringPair evalInterfaceIds
  the evalInterfaceIds aggregate
Friends

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

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

8.119.1 Detailed Description

evaluation id.

ParamResponsePair provides a container class for association of the input for a particular function evaluation (a variables object) with the output from this function evaluation (a response object), along with an evaluation identifier. This container defines the basic unit used in the data_pairs cache, in restart file operations, and in a variety of scheduling algorithm queues. With the advent of STL, replacement of arrays of this class with map<> and pair<> template constructs may be possible (using map<pair<int,String>, pair<Variables,Response>>), for example), assuming that deep copies, I/O, alternate constructors, etc., can be adequately addressed. Boost tuple<> may also be a candidate.

8.119.2 Constructor & Destructor Documentation

8.119.2.1 ParamResponsePair (const Variables & vars, const String & interface_id, const Response & response, bool deep_copy = false) [inline]

alternate constructor for temporaries

Uses of this constructor often employ the standard Variables and Response copy constructors to share representations since this constructor is commonly used for search_pairs (which are local instantiations that go out of scope prior to any changes to values; i.e., they are not used for history).

8.119.2.2 ParamResponsePair (const Variables & vars, const String & interface_id, const Response & response, const int eval_id, bool deep_copy = true) [inline]

standard constructor for history uses

Uses of this constructor often do not share representations since deep copies are used when history mechanisms (e.g., data_pairs and beforeSynchCorePRPQueue) are involved.

8.119.3 Member Function Documentation

8.119.3.1 void read (MPIUnpackBuffer & s) [inline]

read a ParamResponsePair object from a packed MPI buffer

idInterface is omitted since master processor retains interface ids and communicates asv and response data only with slaves.
8.119.3.2 void write (MPIPackBuffer & s) const [inline]

write a ParamResponsePair object to a packed MPI buffer
idInterface is omitted since master processor retains interface ids and communicates asv and response data only
with slaves.

8.119.4 Member Data Documentation

8.119.4.1 IntStringPair evalInterfaceIds [private]

the evalInterfaceIds aggregate
the function evaluation identifier (assigned from ApplicationInterface::fnEvalId) is paired with the interface used
to generate the response object. Used in PRPCache id_vars_set_compare to prevent duplicate detection on results
from different interfaces. evalInterfaceIds belongs here rather than in Response since some Response objects
involve consolidation of several fn evals (e.g., Model::synchronize_derivatives()) that are not, in total, generated
by a single interface. The prPair, on the other hand, is used for storage of all low level fn evals that get evaluated
in ApplicationInterface::map().

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

- ParamResponsePair.H
- ParamResponsePair.C
8.120  ParamStudy Class Reference

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

Inheritance diagram for ParamStudy::

```
  Iterator
     |    
  Analyzer
     |    
PStudyDACE
     |    
  ParamStudy
```

Public Member Functions

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

- **∼ParamStudy** ()

  *destructor*

- void **pre_run** ()

  *pre-run portion of run_iterator (optional)*

- void **extract_trends** ()

  *Redefines the run_iterator virtual function for the PStudy/DACE branch.*

- void **derived_post_run** ()

  *portions of post_run specific to derived iterators*

Private Member Functions

- void **sample** ()

  *performs the parameter study by sampling from a list of points*

- void **vector_loop** ()

  *step vectors*

- void **centered_loop** ()

  *centered about an initial point*
- void multidim_loop ()
  defined by a set of multidimensional partitions

- bool distribute_list_of_points (const RealVector &list_of_pts)
  and listDRVPoints

- bool distribute_step_vector (const RealVector &step_vector)
  distributes incoming step_vector among contStepVector and discStepVector

- void final_point_to_step_vector ()
  compute step vectors from finalPoint, initial points, and numSteps

- void distribute_partitions ()
  compute step vectors from variablePartitions and global bounds

- bool check_num_steps (int num_steps)
  perform error checks on numSteps

- bool check_final_point (const RealVector &final_pt)
  perform error checks on finalPoint

- bool check_steps_per_variable (const IntVector &steps_per_var)
  perform error checks on stepsPerVariable

- bool check_variable_partitions (const UShortArray &partitions)
  perform error checks on variablePartitions

- bool check_finite_bounds ()
  as required for computing partitions of finite ranges

- bool check_ranges_sets (int num_steps)
  sanity check for vector parameter study

- bool check_ranges_sets (const IntVector &steps)
  sanity check for centered parameter study

- bool check_sets (const IntVector &steps)
  sanity check for increments along int/real set dimensions

- int truncate (const Real &value) const
  cast Real to int and ensure no resulting change in value

- int integer_step (int range, int num_steps) const
  check for integer remainder and return step

- int index_step (size_t start, size_t end, int num_steps) const
check for out of bounds and index remainder and return step

- **void write_ordered**(std::ostream &s, const RealVector &c_vector, const IntVector &di_vector, const RealVector &dr_vector)
  reorder CV/DIV/DRV into standard output order

- **void write_ordered**(std::ostream &s, const RealVector &c_vector, const IntVector &d_vector)
  reorder CV/DV into standard output order

- **void c_step**(size_t c_index, int increment, Variables &vars)
  helper function for performing a continuous step in one variable

- **void dri_step**(size_t d_index, size_t di_index, int increment, Variables &vars)
  range variable

- **void dsi_step**(size_t d_index, size_t di_index, int increment, const IntSet &values, Variables &vars)
  helper function for performing a discrete step in an integer set variable

- **void dsr_step**(size_t d_index, size_t dr_index, int increment, const RealSet &values, Variables &vars)
  helper function for performing a discrete step in a real set variable

### Private Attributes

- **short pStudyType**
  CENTERED, or MULTIDIM.

- **size_t numEvals**
  total number of parameter study evaluations computed from specification

- **RealVectorArray listCVPoints**
  array of continuous evaluation points for the list_parameter_study

- **IntVectorArray listDIVPoints**
  array of discrete int evaluation points for the list_parameter_study

- **RealVectorArray listDRVPoints**
  array of discrete real evaluation points for the list_parameter_study

- **RealVector initialCVPoint**
  the continuous starting point for vector and centered parameter studies

- **IntVector initialDIVPoint**
  the continuous starting point for vector and centered parameter studies

- **RealVector initialDRVPoint**
the continuous starting point for vector and centered parameter studies

- RealVector finalPoint
  the ending point for vector_parameter_study (a specification option)

- RealVector contStepVector
  the n-dimensional continuous increment in vector_parameter_study

- IntVector discStepVector
  the n-dimensional discrete increment in vector_parameter_study

- int numSteps
  the number of times stepVector is applied in vector_parameter_study

- IntVector stepsPerVariable
  variable in a centered_parameter_study

- UShortArray variablePartitions
  number of partitions for each variable in a multidim_parameter_study

### 8.120.1 Detailed Description

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

The ParamStudy class contains several algorithms for performing parameter studies of different types. The vector parameter study steps along an n-dimensional vector from an arbitrary initial point to an arbitrary final point in a specified number of steps. The centered parameter study performs a number of plus and minus offsets in each coordinate direction around a center point. A multidimensional parameter study fills an n-dimensional hypercube based on bounds and a specified number of partitions for each dimension. And the list parameter study provides for a user specification of a list of points to evaluate, which allows general parameter investigations not fitting the structure of vector, centered, or multidim parameter studies.

### 8.120.2 Member Function Documentation

#### 8.120.2.1 void pre_run () [virtual]

pre-run portion of run_iterator (optional)

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely contained in the derived run function

Reimplemented from Iterator.

#### 8.120.2.2 void derived_post_run () [virtual]

portions of post_run specific to derived iterators
Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual derived class portion of post_run(). Redefinition by derived classes is optional.

Reimplemented from Iterator.

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

- ParamStudy.H
- ParamStudy.C
8.121 partial_prp_equality Struct Reference

predicate for comparing ONLY the idInterface and Vars attributes of PRPair

Public Member Functions

- bool operator() (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr) const

8.121.1 Detailed Description

predicate for comparing ONLY the idInterface and Vars attributes of PRPair

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

- PRPMultiIndex.H
8.122  **partial_prp_hash** Struct Reference

wrapper to delegate to the `ParamResponsePair hash_value` function

**Public Member Functions**

- `std::size_t operator() (const ParamResponsePair &prp) const`

8.122.1  **Detailed Description**

wrapper to delegate to the `ParamResponsePair hash_value` function

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

- `PRPMultiIndex.H`
8.123 ProblemDescDB Class Reference

The database containing information parsed from the DAKOTA input file.

Inheritance diagram for ProblemDescDB::

```
ProblemDescDB
  NIDRProblemDescDB
```

Public Member Functions

- **ProblemDescDB ()**
  
  *default constructor*

- **ProblemDescDB (ParallelLibrary &parallel_lib)**
  
  *standard constructor*

- **ProblemDescDB (const ProblemDescDB &db)**
  
  *copy constructor*

- **~ProblemDescDB ()**
  
  *destructor*

- **ProblemDescDB operator= (const ProblemDescDB &db)**
  
  *assignment operator*

- **void manage_inputs (CommandLineHandler &cmd_line_handler)**
  
  *normal API employed in main.C.*

- **void manage_inputs (const char *dakota_input_file, const char *parser_options=NULL, void(*callback)(void *)=NULL, void *callback_data=NULL)**
  
  *library_mode.C.*

- **void parse_inputs (const char *dakota_input_file, const char *parser_options=NULL, void(*callback)(void *)=NULL, void *callback_data=NULL)**
  
  *have been provided.*

- **void check_input ()**
  
  *keywords in the dakota input file. Used by parse_inputs().*

- **void broadcast ()**
  
  *data across the processor allocation. Used by manage_inputs().*
• void post_process ()
  variables/responses specification arrays. Used by manage_inputs().

• void lock ()
  may not be set properly. Unlocked by a set nodes operation.

• void unlock ()
  Explicitly unlocks the database. Use with care.

• void set_db_list_nodes (const String &method_tag)
  this method specification to set all other list iterators.

• void set_db_list_nodes (const size_t &method_index)
  specification to set all other list iterators.

• void resolve_top_method ()
  to the top method and then sets the list nodes accordingly.

• void set_db_method_node (const String &method_tag)
  particular method specification (only).

• void set_db_method_node (const size_t &method_index)
  particular method specification (only).

• size_t get_db_method_node ()
  return the index of the active node in dataMethodList

• void set_db_model_nodes (const String &model_tag)
  identifier string

• void set_db_model_nodes (const size_t &model_index)
  within dataModelList

• size_t get_db_model_node ()
  return the index of the active node in dataModelList

• void set_db_variables_node (const String &variables_tag)
  set dataVariablesIter based on the variables identifier string

• void set_db_interface_node (const String &interface_tag)
  set dataInterfaceIter based on the interface identifier string

• void set_db_responses_node (const String &responses_tag)
  set dataResponsesIter based on the responses identifier string

• ParallelLibrary & parallel_library () const
return the parallelLib reference

- **IteratorList** & **iterator_list** ()
  return a list of all **Iterator** objects that have been instantiated

- **ModelList** & **model_list** ()
  return a list of all **Model** objects that have been instantiated

- **VariablesList** & **variables_list** ()
  return a list of all **Variables** objects that have been instantiated

- **InterfaceList** & **interface_list** ()
  return a list of all **Interface** objects that have been instantiated

- **ResponseList** & **response_list** ()
  return a list of all **Response** objects that have been instantiated

- const **RealVector** & **get_rdv** (const **String** &entry_name) const
  get a **RealVector** out of the database based on an identifier string

- const **IntVector** & **get_idv** (const **String** &entry_name) const
  get an **IntVector** out of the database based on an identifier string

- const **UShortArray** & **get_dusa** (const **String** &entry_name) const
  get a **UShortArray** out of the database based on an identifier string

- const **RealSymMatrix** & **get_rsdm** (const **String** &entry_name) const
  get a **RealSymMatrix** out of the database based on an identifier string

- const **RealVectorArray** & **get_rdva** (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 an **IntList** out of the database based on an identifier string

- const **IntSet** & **get_dis** (const **String** &entry_name) const
  get an **IntSet** out of the database based on an identifier string

- const **IntSetArray** & **get_disa** (const **String** &entry_name) const
  get an **IntSetArray** out of the database based on an identifier string

- const **RealSetArray** & **get_drsa** (const **String** &entry_name) const
  get a **RealSetArray** 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

• **int get_int** (const String &entry_name) const

  get an int out of the database based on an identifier string

• **short get_short** (const String &entry_name) const

  get a short out of the database based on an identifier string

• **unsigned short get_ushort** (const String &entry_name) const

  get an unsigned short out of the database based on an identifier string

• **size_t get_sizet** (const String &entry_name) const

  get a size_t out of the database based on an identifier string

• **bool get_bool** (const String &entry_name) const

  get a bool out of the database based on an identifier string

• **void insert_node** (const DataStrategy &data_strategy)

  set the DataStrategy object

• **void insert_node** (const DataMethod &data_method)

  add a DataMethod object to the dataMethodList

• **void insert_node** (const DataModel &data_model)

  add a DataModel object to the dataModelList

• **void insert_node** (DataVariables &data_variables)

  add a DataVariables object to the dataVariablesList

• **void insert_node** (const DataInterface &data_interface)

  add a DataInterface object to the dataInterfaceList

• **void insert_node** (const DataResponses &data_responses)

  add a DataResponses object to the dataResponsesList

• **void set** (const String &entry_name, const RealVector &rdv)

  set a RealVector within the database based on an identifier string
• void set (const String &entry_name, const IntVector &idv)
  set an IntVector within the database based on an identifier string

• void set (const String &entry_name, const RealSymMatrix &rsdm)
  set a RealMatrix within the database based on an identifier string

• void set (const String &entry_name, const RealVectorArray &rdva)
  identifier string

• void set (const String &entry_name, const StringArray &dsa)
  set a StringArray within the database based on an identifier string

• bool is_null () const
  function to check dbRep (does this envelope contain a letter)

Protected Member Functions

• ProblemDescDB (BaseConstructor, ParallelLibrary &parallel_lib)
  derived class constructors - Coplien, p. 139)

• virtual void derived_parse_inputs (const char *dakota_input_file, const char *parser_options)
  derived class specifics within parse_inputs()

• virtual void derived_broadcast ()
  derived class specifics within broadcast()

• virtual void derived_post_process ()
  derived class specifics within post_process()

Protected Attributes

• DataStrategy strategySpec
  to strategy_kwhandler() or insert_node()

• List< DataMethod > dataMethodList
  or insert_node()

• List< DataModel > dataModelList
  or insert_node()

• List< DataVariables > dataVariablesList
  variables_kwhandler() or insert_node()
DAKOTA Class Documentation

- **List**< DataInterface > dataInterfaceList
  
  interface_kwhandler() or insert_node()

- **List**< DataResponses > dataResponsesList
  
  responses_kwhandler() or insert_node()

- size_t strategyCntr
  
  counter for strategy specifications used in check_input

**Private Member Functions**

- const **Iterator** & get_iterator (Model &model)
  
  retrieve an existing Iterator, if it exists, or instantiate a new one

- const **Model** & get_model ()
  
  retrieve an existing Model, if it exists, or instantiate a new one

- const **Variables** & get_variables ()
  
  retrieve an existing Variables, if it exists, or instantiate a new one

- const **Interface** & get_interface ()
  
  retrieve an existing Interface, if it exists, or instantiate a new one

- const **Response** & get_response (const **Variables** &vars)
  
  retrieve an existing Response, if it exists, or instantiate a new one

- **ProblemDescDB** * get_db (ParallelLibrary &parallel_lib)
  
  Used by the envelope constructor to instantiate the correct letter class.

- void send_db_buffer ()
  
  and dataResponsesList. Used by manage_inputs().

- void receive_db_buffer ()
  
  and dataResponsesList. Used by manage_inputs().

**Private Attributes**

- ParallelLibrary & parallelLib
  
  reference to the parallel_lib object passed from main

- **List**< DataMethod >::iterator dataMethodIter
  
  iterator identifying the active list node in dataMethodList
- List< DataModel >::iterator dataModelIter
  iterator identifying the active list node in dataModelList

- List< DataVariables >::iterator dataVariablesIter
  iterator identifying the active list node in dataVariablesList

- List< DataInterface >::iterator dataInterfaceIter
  iterator identifying the active list node in dataInterfaceList

- List< DataResponses >::iterator dataResponsesIter
  iterator identifying the active list node in dataResponsesList

- IteratorList iteratorList
  list of iterator objects, one for each method specification

- ModelList modelList
  list of model objects, one for each model specification

- VariablesList variablesList
  list of variables objects, one for each variables specification

- InterfaceList interfaceList
  list of interface objects, one for each interface specification

- ResponseList responseList
  list of response objects, one for each responses specification

- bool methodDBLocked
  prior to setting the list node for the active method specification

- bool modelDBLocked
  prior to setting the list node for the active model specification

- bool variablesDBLocked
  prior to setting the list node for the active variables specification

- bool interfaceDBLocked
  prior to setting the list node for the active interface specification

- bool responsesDBLocked
  prior to setting the list node for the active responses specification

- ProblemDescDB * dbRep
  pointer to the letter (initialized only for the envelope)

- int referenceCount
  number of objects sharing dbRep
Friends

- class Model
  
  Model requires access to get_variables() and get_response().

- class SingleModel
  
  SingleModel requires access to get_interface().

- class HierarchSurrModel
  
  HierarchSurrModel requires access to get_model().

- class DataFitSurrModel
  
  DataFitSurrModel requires access to get_iterator() and get_model().

- class NestedModel
  
  get_iterator(), and get_model().

- class Strategy
  
  Strategy requires access to get_iterator().

- class SingleMethodStrategy
  
  SingleMethodStrategy requires access to get_model().

- class HybridStrategy
  
  HybridStrategy requires access to get_model().

- class ConcurrentStrategy
  
  ConcurrentStrategy requires access to get_model().

- class SurrBasedLocalMinimizer
  
  SurrBasedLocalMinimizer requires access to get_iterator().

- class SurrBasedGlobalMinimizer
  
  SurrBasedGlobalMinimizer requires access to get_iterator().

8.123.1 Detailed Description

The database containing information parsed from the DAKOTA input file.

The ProblemDescDB class is a database for DAKOTA input file data that is populated by a parser defined in a derived class. When the parser reads a complete keyword, it populates a data class object (DataStrategy, DataMethod, DataVariables, DataInterface, or DataResponses) and, for all cases except strategy, appends the object to a linked list (dataMethodList, dataVariablesList, dataInterfaceList, or dataResponsesList). No strategy linked list is used since only one strategy specification is allowed.
8.123 ProblemDescDB Class Reference

8.123.2 Constructor & Destructor Documentation

8.123.2.1 ProblemDescDB ()

default constructor
The default constructor: dbRep is NULL in this case. This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

8.123.2.2 ProblemDescDB (ParallelLibrary & parallel_lib)

standard constructor
This is the envelope constructor which uses problem_db to build a fully populated db object. It only needs to extract enough data to properly execute get_db(problem_db), since the constructor overloaded with Base-Constructor builds the actual base class data inherited by the derived classes.

8.123.2.3 ProblemDescDB (const ProblemDescDB & db)

copy constructor
Copy constructor manages sharing of dbRep and incrementing of referenceCount.

8.123.2.4 ~ProblemDescDB ()

destructor
Destructor decrements referenceCount and only deletes dbRep when referenceCount reaches zero.

8.123.2.5 ProblemDescDB (BaseConstructor, ParallelLibrary & parallel_lib) [protected]
derived class constructors - Coplien, p. 139
This constructor is the one which must build the base class data for all derived classes. get_db() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_db() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~ProblemDescDB).

8.123.3 Member Function Documentation

8.123.3.1 ProblemDescDB operator= (const ProblemDescDB & db)

assignment operator
8.123.3.2 void manage_inputs (CommandLineHandler & cmd_line_handler)

normal API employed in main.C.
Manage command line inputs using the CommandLineHandler class and parse the input file.

8.123.3.3 void manage_inputs (const char * dakota_input_file, const char * parser_options = NULL,
void (*)(void *) callback = NULL, void * callback_data = NULL)

library_mode.C.
Parse the input file, broadcast it to all processors, and post-process the data on all processors.

8.123.3.4 void parse_inputs (const char * dakota_input_file, const char * parser_options = NULL,
void (*)(void *) callback = NULL, void * callback_data = NULL)

have been provided.
Parse the input file, execute the callback function (if present), and perform basic checks on keyword counts.

8.123.3.5 void post_process ()

variables/responses specification arrays. Used by manage_inputs().
When using library mode in a parallel application, post_process() should be called on all processors following broadcast() of a minimal problem specification.

8.123.3.6 ProblemDescDB * get_db (ParallelLibrary & parallel_lib) [private]

Used by the envelope constructor to instantiate the correct letter class.
Initializes dbRep to the appropriate derived type. The standard derived class constructors are invoked.
The documentation for this class was generated from the following files:

- ProblemDescDB.H
- ProblemDescDB.C
8.124 PStudyDACE Class Reference

design of experiments methods.

Inheritance diagram for PStudyDACE:

```
Protected Member Functions

- PStudyDACE (Model &model)
  constructor

- PStudyDACE (NoDBBaseConstructor, Model &model)
  alternate constructor for instantiations "on the fly"

- ~PStudyDACE ()
  destructor

- void run ()
  and may contain pre/post steps in lieu of separate pre/post

- void print_results (std::ostream &s)
  print the final iterator results

- virtual void extract_trends ()=0
  Redefines the run_iterator virtual function for the PStudy/DACE branch.

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

Protected Attributes

- SensAnalysisGlobal pStudyDACESensGlobal
  initialize statistical post processing

- bool volQualityFlag
```
flag which specifies evaluation of volumetric quality measures

Private Attributes

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

8.124.1 Detailed Description

design of experiments methods.
The PStudyDACE base class manages common data and functions, such as those involving the best solutions located during the parameter set evaluations or the printing of final results.

8.124.2 Member Function Documentation

8.124.2.1 void run () [inline, protected, virtual]

and may contain pre/post steps in lieu of separate pre/post
Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.

8.124.2.2 void print_results (std::ostream & s) [protected, virtual]

print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
Reimplemented from Analyzer.
8.124.2.3 void volumetric_quality (int ndim, int num_samples, double * sample_points) [protected]

Calculation of volumetric quality measures.
Calculation of volumetric quality measures developed by FSU.
The documentation for this class was generated from the following files:

- DakotaPStudyDACE.H
- DakotaPStudyDACE.C
8.125 **PSUADEDesignCompExp Class Reference**

Wrapper class for the PSUADE library.

Inheritance diagram for PSUADEDesignCompExp:

```
  Iterator
   |     |
   |     |
   Analyzer
   |     |
   PStudyDACE
   |     |
  PSUADEDesignCompExp
```

**Public Member Functions**

- **PSUADEDesignCompExp (Model &model)**
  
  *primary constructor for building a standard DACE iterator*

- **~PSUADEDesignCompExp ()**
  
  *destructor*

- **void pre_run ()**
  
  *pre-run portion of run_iterator (optional)*

- **void extract_trends ()**
  
  *Redefines the run_iterator virtual function for the PStudy/DACE branch.*

- **void derived_post_run ()**
  
  *portions of post_run specific to derived iterators*

- **void sampling_reset (int min_samples, int rec_samples, bool all_data_flag, bool stats_flag)**
  
  *reset sampling iterator*

- **const String & sampling_scheme () const**
  
  *return sampling name*

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

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

- void enforce_input_rules ()
  
  enforce sanity checks/modifications for the user input specification

Private Attributes

- int samplesSpec
  
  initial specification of number of samples

- int numSamples
  
  current number of samples to be evaluated

- const UShortArray & varPartitionsSpec
  
  number of partitions in each variable direction

- int numPartitions
  
  number of partitions to pass to PSUADE (levels = partitions + 1)

- bool allDataFlag
  
  Iterator::all_variables() and Iterator::all_responses().

- size_t numDACERuns
  
  counter for number of run() executions for this object

- bool varyPattern
  
  but are still repeatable

- const int seedSpec
  
  (allows repeatable results)

- int randomSeed
  
  current seed for the random number generator

8.125.1 Detailed Description

Wrapper class for the PSUADE library.

The PSUADEDesignCompExp class provides a wrapper for PSUADE, a C++ design of experiments library from Lawrence Livermore National Laboratory. Currently this class only includes the PSUADE Morris One-at-a-time (MOAT) method to uniformly sample the parameter space spanned by the active bounds of the current Model. It returns all generated samples and their corresponding responses as well as the best sample found.
8.125.2 Constructor & Destructor Documentation

8.125.2.1 PSUADEDesignCompExp (Model & model)

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

8.125.3 Member Function Documentation

8.125.3.1 void pre_run () [virtual]

pre-run portion of run_iterator (optional)
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely contained
in the derived run function
Reimplemented from Iterator.

8.125.3.2 void derived_post_run () [virtual]

portions of post_run specific to derived iterators
Iterator supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual de-
ived class portion of post_run(). Redefinition by derived classes is optional.
Reimplemented from Iterator.

8.125.3.3 void enforce_input_rules () [private]

enforce sanity checks/modifications for the user input specification
Users may input a variety of quantities, but this function must enforce any restrictions imposed by the sampling
algorithms.
The documentation for this class was generated from the following files:

- PSUADEDesignCompExp.H
- PSUADEDesignCompExp.C
8.126 RecastBaseConstructor Struct Reference

instantiations.

Public Member Functions

- RecastBaseConstructor (int=0)

  C++ structs can have constructors.

8.126.1 Detailed Description

instantiations.

RecastBaseConstructor is used to overload the constructor used for on-the-fly Model instantiations. Putting this struct here avoids circular dependencies.

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

- global_defs.h
8.127 **RecastModel Class Reference**

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

Inheritance diagram for RecastModel::

```
Model

RecastModel
```

**Public Member Functions**


  standard constructor

- `RecastModel (Model &sub_model, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset)`

  alternate constructor

- `~RecastModel ()`

  destructor


  completes initialization of the RecastModel after alternate construction

- `void submodel_supports_estimated_derivatives (bool ssed_flag)`

  override the submodel’s derivative estimation behavior
Protected Member Functions

- `void derived_compute_response (const ActiveSet &set)`
  
  (forward to subModel.compute_response())

- `void derived_asynch_compute_response (const ActiveSet &set)`
  
  (forward to subModel.asynch_compute_response())

- `const IntResponseMap & derived_synchronize ()`
  
  (forward to subModel.synchronize())

- `const IntResponseMap & derived_synchronize_nowait ()`
  
  (forward to subModel.synchronize_nowait())

- `Iterator & subordinate_iterator ()`
  
  return sub-iterator, if present, within subModel

- `Model & subordinate_model ()`
  
  return subModel

- `Model & surrogate_model ()`
  
  return surrogate model, if present, within subModel

- `Model & truth_model ()`
  
  return truth model, if present, within subModel

- `void derived_subordinate_models (ModelList &ml, bool recurse_flag)`
  
  add subModel to list and recurse into subModel

- `void update_from_subordinate_model (bool recurse_flag=true)`
  
  pass request to subModel if recursing and then update from it

- `Interface & interface ()`
  
  return subModel interface

- `void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)`
  
  squares terms and optionally recurses into subModel

- `void surrogate_function_indices (const IntSet &surr_fn_indices)`
  
  forward to subModel

- `void surrogate_bypass (bool bypass_flag)`
  
  models contained within this model

- `void build_approximation ()`
  
  builds the subModel approximation
• bool build_approximation (const Variables &vars, const Response &response)
  builds the subModel approximation

• void update_approximation (const Variables &vars, const Response &response, bool rebuild_flag)
  updates the subModel approximation

• void update_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  updates the subModel approximation

• void append_approximation (const Variables &vars, const Response &response, bool rebuild_flag)
  appends the subModel approximation

• void append_approximation (const VariablesArray &vars_array, const ResponseArray &resp_array, bool rebuild_flag)
  appends the subModel approximation

• Array< Approximation > & approximations ()
  retrieve the set of Approximations from the subModel

• const RealVectorArray & approximation_coefficients ()
  retrieve the approximation coefficients from the subModel

• void approximation_coefficients (const RealVectorArray &approx_coeffs)
  set the approximation coefficients within the subModel

• void print_coefficients (std::ostream &s, size_t index) const
  print a particular set of approximation coefficients within the subModel

• const RealVector & approximation_variances (const RealVector &c_vars)
  retrieve the approximation variances from the subModel

• const List< SurrogateDataPoint > & approximation_data (size_t index)
  retrieve the approximation data from the subModel

• void component_parallel_mode (short mode)
  virtual function redefinition is simply a sanity check.

• String local_eval_synchronization ()
  return subModel local synchronization setting

• int local_eval_concurrency ()
  return subModel local evaluation concurrency

• bool derived_master_overload () const
8.127 RecastModel Class Reference

- **evaluation** (request forwarded to subModel)

  - void **derived_init_communicators** (const int &max_iterator_concurrency, bool recurse_flag=true)
    set up RecastModel for parallel operations (request forwarded to subModel)

  - void **derived_init_serial** ()
    set up RecastModel for serial operations (request forwarded to subModel).

  - void **derived_set_communicators** (const int &max_iterator_concurrency, bool recurse_flag=true)
    set active parallel configuration within subModel

  - void **derived_free_communicators** (const int &max_iterator_concurrency, bool recurse_flag=true)
    to subModel

  - void **serve** ()
    Completes when a termination message is received from stop_servers().

  - void **stop_servers** ()
    when RecastModel iteration is complete.

  - void **inactive_view** (short view, bool recurse_flag=true)
    context and optionally recurse into subModel

  - const **String & interface_id** () const
    return the subModel interface identifier

  - int **evaluation_id** () const
    forwarded to subModel

  - void **set_evaluation_reference** ()
    (request forwarded to subModel)

  - void **fine_grained_evaluation_counters** ()
    request fine-grained evaluation reporting within subModel

  - void **print_evaluation_summary** (std::ostream &s, bool minimal_header=false, bool relative_count=true)
    const
    forwarded to subModel

**Private Member Functions**

  - void **set_mapping** (const ActiveSet &recast_set, ActiveSet &sub_model_set)
    into sub_model_set for use with subModel.

  - void **update_from_sub_model** ()
    update current variables/labels/bounds/targets from subModel
Private Attributes

- Model subModel
  the sub-model underlying the function pointers

- Sizet2DArray varsMapIndices
  subModel variables

- bool nonlinearVarsMapping
  Hessians are managed per function, not per variable.

- bool respMapping
  are supplied

- Sizet2DArray primaryRespMapIndices
  to RecastModel Response).

- Sizet2DArray secondaryRespMapIndices
  to RecastModel response).

- BoolDequeArray nonlinearRespMapping
  augment the subModel function value/gradient requirements.

- IntActiveSetMap recastSetMap
  Needed for currentResponse update in synchronization routines.

- IntVariablesMap recastVarsMap
  synchronization routines.

- IntVariablesMap subModelVarsMap
  synchronization routines.

- IntResponseMap recastResponseMap
  and RecastModel::derived_synchronize_nowait()

- void(* variablesMapping )(const Variables &recast_vars, Variables &sub_model_vars)
  holds pointer for variables mapping function passed in ctor/initialize

- void(* setMapping )(const ActiveSet &recast_set, ActiveSet &sub_model_set)
  holds pointer for set mapping function passed in ctor/initialize

- void(* primaryRespMapping )(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  ctor/initialize

8.127 RecastModel Class Reference

8.127.1 Detailed Description

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

The RecastModel class uses function pointers to allow recasting of the subModel input/output into new problem forms. This is currently used to recast SBO approximate subproblems, but can be used for multiobjective, input/output scaling, and other problem modifications in the future.

8.127.2 Constructor & Destructor Documentation

8.127.2.1 RecastModel (Model & sub_model, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset)

alternate constructor

This alternate constructor defers initialization of the function pointers until a separate call to initialize(), and accepts the minimum information needed to construct currentVariables, currentResponse, and userDefinedConstraints. The resulting model is sufficiently complete for passing to an Iterator.

8.127.3 Member Function Documentation

8.127.3.1 void initialize (const Sizet2DArray & vars_map_indices, bool nonlinear_vars_mapping, void(*)(const Variables & recast_vars, Variables & sub_model_vars) variables_map, void(*)(const ActiveSet & recast_set, ActiveSet & sub_model_set) set_map, const Sizet2DArray & primary_resp_map_indices, const Sizet2DArray & secondary_resp_map_indices, const BoolDequeArray & nonlinear_resp_mapping, void(*)(const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) primary_resp_map, void(*)(const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) secondary_resp_map)

completes initialization of the RecastModel after alternate construction

This function is used for late initialization of the recasting functions. It is used in concert with the alternate constructor.

8.127.3.2 void update_from_sub_model () [private]

update current variables/labels/bounds/targets from subModel

Update inactive values and labels in currentVariables and inactive bound constraints in userDefinedConstraints from variables and constraints data within subModel.

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

- RecastModel.H
- RecastModel.C

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
8.128 Response Class Reference

Response provides the handle class.

Public Member Functions

- **Response ()**
  
  *default constructor*

- **Response (const Variables &vars, const ProblemDescDB &problem_db)**
  
  *standard constructor built from problem description database*

- **Response (const ActiveSet &set)**
  
  *alternate constructor using limited data*

- **Response (const Response &response)**
  
  *copy constructor*

- **∼Response ()**
  
  *destructor*

- **Response operator= (const Response &response)**
  
  *assignment operator*

- **size_t num_functions () const**
  
  *return the number of response functions*

- **const ActiveSet & active_set () const**
  
  *return the active set*

- **void active_set (const ActiveSet &set)**
  
  *set the active set*

- **const ShortArray & active_set_request_vector () const**
  
  *return the active set request vector*

- **void active_set_request_vector (const ShortArray &asrv)**
  
  *set the active set request vector*

- **const UIntArray & active_set_derivative_vector () const**
  
  *return the active set derivative vector*

- **void active_set_derivative_vector (const UIntArray &asdv)**
  
  *set the active set derivative vector*
- `const String & responses_id() const
  return the response identifier

- `const String & function_label(const size_t &i) const
  return a response function identifier string

- `const StringArray & function_labels() const
  return the response function identifier strings

- `void function_label(const String & label, const size_t &i)
  set a response function identifier string

- `void function_labels(const StringArray & labels)
  set the response function identifier strings

- `const Real & function_value(const size_t &i) const
  return a function value

- `const RealVector & function_values() const
  return all function values

- `void function_value(const Real & function_val, const size_t &i)
  set a function value

- `void function_values(const RealVector & function_vals)
  set all function values

- `RealVector function_gradient(const int &i) const
  return a function gradient as a Teuchos_SerialDenseVector VIEW

- `RealVector function_gradient_copy(const int &i) const
  return a function gradient as a Teuchos::Copy vector (deep copy)

- `const RealMatrix & function_gradients() const
  return all function gradients

- `void function_gradient(const RealVector & function_grad, const int &i)
  set a function gradient

- `void function_gradients(const RealMatrix & function_grads)
  set all function gradients

- `const RealSymMatrix & function_hessian(const size_t &i) const
  return a function Hessian

- `const RealSymMatrixArray & function_hessians() const
  return all function Hessians
return all function Hessians

- void function_hessian (const RealSymMatrix &function_hessian, const size_t &i)
  set a function Hessian

- void function_hessians (const RealSymMatrixArray &function_hessians)
  set all function Hessians

- void read (std::istream &s)
  read a response object from an std::istream

- void write (std::ostream &s) const
  write a response object to an std::ostream

- void read_annotated (std::istream &s)
  read a response object in annotated format from an std::istream

- void write_annotated (std::ostream &s) const
  write a response object in annotated format to an std::ostream

- void read_tabular (std::istream &s)
  read responseRep::functionValues in tabular format from an std::istream

- void write_tabular (std::ostream &s) const
  write responseRep::functionValues in tabular format to an std::ostream

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

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

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

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

- Response copy () const
  a deep copy for use in history mechanisms

- int data_size ()
  handle class forward to corresponding body class member function

- void read_data (double *response_data)
  handle class forward to corresponding body class member function
• **void write_data** (double *response_data)

  handle class forward to corresponding body class member function

• **void overlay** (const Response &response)

  handle class forward to corresponding body class member function

• **void copy_results** (const Response &response)

  different derivative array sizing between the two response objects.

• **void copy_results** (const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set)

  object. Care is taken to allow different derivative array sizing.

• **void copy_results_partial** (size_t start_index_target, size_t num_items, const Response &response, size_t start_index_source)

  The response objects may have different numbers of response functions.

• **void copy_results_partial** (size_t start_index_target, size_t num_items, const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set, size_t start_index_source)

  of response functions.

• **void reshape** (const size_t &num_fns, const size_t &num_params, bool grad_flag, bool hess_flag)

  rehapes response data arrays

• **void reset** ()

  handle class forward to corresponding body class member function

• **void reset_inactive** ()

  handle class forward to corresponding body class member function

• **bool is_null** () const

  function to check responseRep (does this handle contain a body)

### Private Attributes

• **ResponseRep * responseRep**

  pointer to the body (handle-body idiom)
Friends

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

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

8.128.1 Detailed Description

Response provides the handle class.

The Response class is a container class for an abstract set of functions (functionValues) and their first (function-Gradients) and second (functionHessians) derivatives. The functions may involve objective and constraint functions (optimization data set), least squares terms (parameter estimation data set), or generic response functions (uncertainty quantification data set). It is not currently part of a class hierarchy, since the abstraction has been sufficiently general and has not required specialization. For memory efficiency, it employs the "handle-body idiom" approach to reference counting and representation sharing (see Coplien "Advanced C++", p. 58), for which Response serves as the handle and ResponseRep serves as the body.

8.128.2 Constructor & Destructor Documentation

8.128.2.1 Response ()

default constructor

Need a populated problem description database to build a meaningful Response object, so set the response-Rep=NULL in default constructor for efficiency. This then requires a check on NULL in the copy constructor, assignment operator, and destructor.

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

- DakotaResponse.H
- DakotaResponse.C


8.129  **ResponseRep Class Reference**

*ResponseRep* provides the body class.

**Private Member Functions**

- **ResponseRep ()**
  
  *default constructor*

- **ResponseRep (const Variables &vars, const ProblemDescDB &problem_db)**
  
  *standard constructor built from problem description database*

- **ResponseRep (const ActiveSet &set)**
  
  *alternate constructor using limited data*

- **~ResponseRep ()**
  
  *destructor*

- **void read (std::istream &s)**
  
  *read a responseRep object from an std::istream*

- **void write (std::ostream &s) const**
  
  *write a responseRep object to an std::ostream*

- **void read_annotated (std::istream &s)**
  
  *read a responseRep object from an std::istream (annotated format)*

- **void write_annotated (std::ostream &s) const**
  
  *write a responseRep object to an std::ostream (annotated format)*

- **void read_tabular (std::istream &s)**
  
  *read functionValues from an std::istream (tabular format)*

- **void write_tabular (std::ostream &s) const**
  
  *write functionValues to an std::ostream (tabular format)*

- **void read (BiStream &s)**
  
  *read a responseRep object from a binary stream*

- **void write (BoStream &s) const**
  
  *write a responseRep object to a binary stream*

- **void read (MPIUnpackBuffer &s)**
  
  *read a responseRep object from a packed MPI buffer*
• void write (MPIPackBuffer &s) const
  write a responseRep object to a packed MPI buffer

• int data_size ()
  double* response_data arrays passed into read_data and write_data.

• void read_data (double *response_data)
  read from an incoming double* array

• void write_data (double *response_data)
  write to an incoming double* array

• void overlay (const Response &response)
  add incoming response to functionValues/Gradients/Hessians

• void copy_results (const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set)
  update this response object from components of another response object

• void copy_results_partial (size_t start_index_target, size_t num_items, const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set, size_t start_index_source)
  another response object

• void reshape (const size_t &num_fns, const size_t &num_params, bool grad_flag, bool hess_flag)
  reshapes response data arrays

• void reset ()
  resets all response data to zero

• void reset_inactive ()
  resets all inactive response data to zero

• void active_set_request_vector (const ShortArray &asrv)
  of response functions

• void active_set_derivative_vector (const UIntArray &asdv)
  functionGradients/functionHessians if needed

Private Attributes

• int referenceCount
  number of handle objects sharing responseRep

• RealVector functionValues
8.129 ResponseRep Class Reference

abstract set of response functions

- **RealMatrix** `functionGradients`
  first derivatives of the response functions

- **RealSymMatrixArray** `functionHessians`
  second derivatives of the response functions

- **ActiveSet** `responseActiveSet`
  copy of the `ActiveSet` used by the `Model` to generate a `Response` instance

- **StringArray** `functionLabels`
  response function identifiers used to improve output readability

- **String** `idResponses`
  response identifier string from the input file

Friends

- class **Response**
  the handle class can access attributes of the body class directly

- **bool** `operator== (const ResponseRep &rep1, const ResponseRep &rep2)`
  equality operator

8.129.1 Detailed Description

ResponseRep provides the body class.

The `ResponseRep` class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (`Response`) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (`ResponseRep`) actually contains the response data (functionValues, functionGradients, 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).

8.129.2 Constructor & Destructor Documentation

8.129.2.1 **ResponseRep (const Variables & vars, const ProblemDescDB & problem_db)** [private]
standard constructor built from problem description database
The standard constructor used by Dakota::ModelRep.
8.129.2.2 **ResponseRep (const ActiveSet & set)** [private]

alternate constructor using limited data

Used for building a response object of the correct size on the fly (e.g., by slave analysis servers performing execute() on a local_response). functionLabels is not needed for this purpose since it’s not passed in the MPI send/recv buffers. However, NPSOLOptimizer’s user-defined functions option uses this constructor to build bestResponse and bestResponse needs functionLabels for I/O, so construction of functionLabels has been added.

8.129.3 Member Function Documentation

8.129.3.1 **void read (std::istream & s)** [private]

read a responseRep object from an std::istream

ASCII version of read needs capabilities for capturing data omissions or formatting errors (resulting from user error or asynch race condition) and analysis failures (resulting from nonconvergence, instability, etc.).

8.129.3.2 **void write (std::ostream & s) const** [private]

write a responseRep object to an std::ostream

ASCII version of write.

8.129.3.3 **void read.annotated (std::istream & s)** [private]

read a responseRep object from an std::istream (annotated format)

read.annotated() is used for neutral file translation of restart files. Since objects are built solely from this data, annotations are used. This version closely mirrors the BiStream version.

8.129.3.4 **void write.annotated (std::ostream & s) const** [private]

write a responseRep object to an std::ostream (annotated format)

write.annotated() is used for neutral file translation of restart files. Since objects need to be built 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.

8.129.3.5 **void read.tabular (std::istream & s)** [private]

read functionValues from an std::istream (tabular format)

read.tabular is used to read functionValues in tabular format. It is currently only used by ApproximationInterfaces in reading samples from a file. There is insufficient data in a tabular file to build complete response objects; rather, the response object must be constructed a priori and then its functionValues can be set.
8.129.3.6 void write_tabular (std::ostream & s) const [private]

write functionValues to an std::ostream (tabular format)
write_tabular is used for output of functionValues in a tabular format for convenience in post-processing/plotting of DAKOTA results.

8.129.3.7 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.

8.129.3.8 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.

8.129.3.9 void read (MPIUnpackBuffer & s) [private]

read a responseRep object from a packed MPI buffer
UnpackBuffer version differs from BiStream version in the omission of functionLabels. Master processor retains labels and interface ids and communicates asv and response data only with slaves.

8.129.3.10 void write (MPIPackBuffer & s) const [private]

write a responseRep object to a packed MPI buffer
MPIPackBuffer version differs from BoStream version only in the omission of functionLabels. The master processor retains labels and ids and communicates asv and response data only with slaves.

8.129.3.11 void copy_results (const RealVector & source_fn_vals, const RealMatrix & source_fn_grads, const RealSymMatrixArray & source_fn_hessians, const ActiveSet & source_set) [private]

update this response object from components of another response object
Copy function values/gradients/Hessians data _only_. Prevents unwanted overwriting of responseActiveSet, functionLabels, etc. Also, care is taken to account for differences in derivative variable matrix sizing.
8.129.3.12 void copy_results_partial (size_t start_index_target, size_t num_items, const RealVector & source_fn_vals, const RealMatrix & source_fn_grads, const RealSymMatrixArray & source_fn_hessians, const ActiveSet & source_set, size_t start_index_source) [private]

another response object
Copy function values/gradients/Hessians data _only_. Prevents unwanted overwriting of responseActiveSet, functionLabels, etc. Also, care is taken to account for differences in derivative variable matrix sizing.

8.129.3.13 void reshape (const size_t & num_fns, const size_t & num_params, bool grad_flag, bool hess_flag) [private]

reshapes response data arrays
Reshape functionValues, functionGradients, and functionHessians according to num_fns, num_params, grad_flag, and hess_flag.

8.129.3.14 void reset () [private]

resets all response data to zero
Reset all numerical response data (not labels, ids, or active set) to zero.

8.129.3.15 void reset_inactive () [private]

resets all inactive response data to zero
Used to clear out any inactive data left over from previous evaluations.

8.129.4 Member Data Documentation

8.129.4.1 RealMatrix functionGradients [private]

first derivatives of the response functions
the gradient vectors (plural) are column vectors in the matrix (singular) with (row, col) = (variable index, response fn index).

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

- DakotaResponse.H
- DakotaResponse.C
8.130 SensAnalysisGlobal Class Reference

and variance-based decomposition

Public Member Functions

- SensAnalysisGlobal ()
  constructor

- ~SensAnalysisGlobal ()
  destructor

- void compute_correlations (const VariablesArray & vars_samples, const ResponseArray & resp_samples)
  simple, partial, simple rank, and partial rank

- bool correlations_computed () const
  has been invoked

- void print_correlations (std::ostream & s, StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView drv_labels, const StringArray & resp_labels) const
  prints the correlations computed in compute_correlations()

Private Member Functions

- void simple_corr (RealMatrix & total_data, bool rank_on, const int & num_in)
  computes simple correlations

- void partial_corr (RealMatrix & total_data, bool rank_on, const int & num_in)
  computes partial correlations

Static Private Member Functions

- static bool rank_sort (const int & x, const int & y)
  sort algorithm to compute ranks for rank correlations

Private Attributes

- RealMatrix simpleCorr
  matrix to hold simple raw correlations

- RealMatrix simpleRankCorr
matrix to hold simple rank correlations

- RealMatrix partialCorr
  matrix to hold partial raw correlations

- RealMatrix partialRankCorr
  matrix to hold partial rank correlations

- size_t numFs
  number of responses

- size_t numVars
  number of inputs

- bool numericalIssuesRaw
  flag indicating numerical issues in partial raw correlation calculations

- bool numericalIssuesRank
  flag indicating numerical issues in partial rank correlation calculations

- bool corrComputed
  flag indicating whether correlations have been computed

Static Private Attributes

- static RealArray rawData
  array to hold temporary data before sort

8.130.1 Detailed Description

and variance-based decomposition

This class provides code for several of the sampling methods both in the NonD branch and in the PStudyDACE branch. Currently, the utility functions provide global sensitivity analysis through correlation calculations (e.g. simple, partial, rank, raw) as well as variance-based decomposition.

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

- SensAnalysisGlobal.H
- SensAnalysisGlobal.C
8.131 SequentialHybridStrategy Class Reference

models of varying fidelity.

Inheritance diagram for SequentialHybridStrategy::

```
Strategy
  ▼
  HybridStrategy
  ▼
SequentialHybridStrategy
```

Public Member Functions

- SequentialHybridStrategy (ProblemDescDB &problem_db)
  constructor

- ~SequentialHybridStrategy ()
  destructor

Protected Member Functions

- void run_strategy ()
  iterators on different models of varying fidelity

- const Variables & variables_results () const
  return the final solution from selectedIterators (variables)

- const Response & response_results () const
  return the final solution from selectedIterators (response)

- void initialize_iterator (int job_index)
  scheduling function (serve_iterators() or static_schedule_iterators())

- void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
  pack a send_buffer for assigning an iterator job to a server

- void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer)
  unpack a recv_buffer for accepting an iterator job from the scheduler

- void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
  pack a send_buffer for returning iterator results from a server
• void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  unpack a recv_buffer for accepting iterator results from a server

• void update_local_results (int job_index)
  update local prpResults with current iteration results

Private Member Functions

• void run_sequential ()
  run a sequential hybrid

• void run_sequential_adaptive ()
  run a sequential adaptive hybrid

• void partition_results (int job_index, size_t &start_index, size_t &job_size)
  extraction from prpResults

• void extract_parameter_sets (int job_index, VariablesArray &partial_param_sets)
  extract partial_param_sets from prpResults based on job_index

• void extract_results_sets (int job_index, PRPArray &partial_prp_results)
  extract partial_prp_results from prpResults based on job_index

• void merge_results_sets (int job_index, PRPArray &partial_prp_results)
  merge partial_prp_results into prpResults based on job_index

• void update_local_results (PRPArray &partial_prp_results, int job_id)
  update the partial set of final results from the local iterator execution

• void initialize_iterator (const VariablesArray &param_sets)
  initialize_iterator(int) to update the active Model and Iterator

Private Attributes

• String hybridType
  sequential or sequential_adaptive

• size_t seqCount
  hybrid sequence counter: 0 to numIterators-1

• size_t numSolnsTransferred
  to the next iterator
8.131 SequentialHybridStrategy Class Reference

- Real progressMetric
  a sequential adaptive hybrid

- Real progressThreshold
  sequential adaptive hybrid switches to the next method

8.131.1 Detailed Description

models of varying fidelity.

The sequential hybrid minimization strategy has two approaches: (1) the non-adaptive sequential hybrid runs one
method to completion, passes its best results as the starting point for a subsequent method, and continues this
succession until all methods have been executed (the stopping rules are controlled internally by each minimizer),
and (2) the adaptive sequential hybrid uses adaptive stopping rules for the minimizers that are controlled externally
by the strategy. Note that while the strategy is targeted at minimizers, any iterator may be used so long as it defines
the notion of a final solution which can be passed as the starting point for subsequent iterators.

8.131.2 Member Function Documentation

8.131.2.1 void pack_parameters_buffer (MPIPackBuffer & send_buffer, int job_index) [inline, protected, virtual]

pack a send_buffer for assigning an iterator job to a server

This virtual function redefinition is executed on the dedicated master processor for self scheduling. It is not used
for peer partitions.

Reimplemented from Strategy.

8.131.2.2 void unpack_parameters_buffer (MPIUnpackBuffer & recv_buffer) [inline, protected, virtual]

unpack a recv_buffer for accepting an iterator job from the scheduler

This virtual function redefinition is executed on an iterator server for dedicated master self scheduling. It is not
used for peer partitions.

Reimplemented from Strategy.

8.131.2.3 void pack_results_buffer (MPIPackBuffer & send_buffer, int job_index) [inline, protected, virtual]

pack a send_buffer for returning iterator results from a server

This virtual function redefinition is executed either on an iterator server for dedicated master self scheduling or
on peers 2 through n for static scheduling.

Reimplemented from Strategy.
8.131.2.4  void unpack_results_buffer (MPIUnpackBuffer & recv_buffer, int job_index)  [inline, protected, virtual]

unpack a recv_buffer for accepting iterator results from a server
This virtual function redefinition is executed on an strategy master (either the dedicated master processor for self scheduling or peer 1 for static scheduling).
Reimplemented from Strategy.

8.131.2.5  void run_sequential ()  [private]

run a sequential hybrid
In the sequential nonadaptive case, there is no interference with the iterators. Each runs until its own convergence criteria is satisfied. Status: fully operational.

8.131.2.6  void run_sequential_adaptive ()  [private]

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

8.131.2.7  void extract_parameter_sets (int job_index, VariablesArray & partial_param_sets)  [inline, private]

extract partial_param_sets from prpResults based on job_index
This convenience function is executed on an iterator master (static scheduling) or a strategy master (self scheduling) at run initialization time and has access to the full prpResults array (this is All-Reduced for all peers at the completion of each cycle in run_sequential()).

8.131.2.8  void extract_results_sets (int job_index, PRPArray & partial_prp_results)  [inline, private]

extract partial_prp_results from prpResults based on job_index
This convenience function is executed on iterator servers 2 through n (peer partition) following iterator executions and prior to prpResults All-Reduce at bottom of run_sequential(). Therefore, some prpResults entries may be empty.

8.131.2.9  void merge_results_sets (int job_index, PRPArray & partial_prp_results)  [inline, private]

merge partial_prp_results into prpResults based on job_index
This convenience function may be executed on either an iterator server (access to only a partial prpResults array) or the strategy master (access to full prpResults array).
The documentation for this class was generated from the following files:

- SequentialHybridStrategy.H
- SequentialHybridStrategy.C
8.132 SerialDirectApplicInterface Class Reference

plug-ins using assign_rep().

Inheritance diagram for SerialDirectApplicInterface::

```
Interface

ApplicationInterface

DirectApplicInterface

SerialDirectApplicInterface
```

Public Member Functions

- **SerialDirectApplicInterface** (const Dakota::ProblemDescDB &problem_db)
  
  *constructor*

- **~SerialDirectApplicInterface** ()
  
  *destructor*

Protected Member Functions

- **int derived_map_ac**(const Dakota::String &ac_name)
  
  *execute an analysis code portion of a direct evaluation invocation*

8.132.1 Detailed Description

plug-ins using assign_rep().

The plug-in SerialDirectApplicInterface resides in namespace SIM and uses a copy of rosenbrock() to perform serial parameter to response mappings. It may be activated by specifying the –with-plugin configure option, which activates the DAKOTA_PLUGIN macro in dakota_config.h used by main.C (which activates the plug-in code block within that file) and activates the PLUGIN_S declaration defined in Makefile.include and used in Makefile.source (which add this class to the build). Test input files should then use an analysis_driver of "plugin_-rosenbrock".

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

- PluginSerialDirectApplicInterface.H
- PluginSerialDirectApplicInterface.C
8.133 SingleMethodStrategy Class Reference

single model.
Inheritance diagram for SingleMethodStrategy::

```
Strategy
   +--- SingleMethodStrategy
```

Public Member Functions

- **SingleMethodStrategy** (ProblemDescDB &problem_db)
  
  constructor

- **~SingleMethodStrategy** ()
  
  destructor

- **void run_strategy** ()
  
  Perform the strategy by executing selectedIterator on userDefinedModel.

- **const Variables & variables_results** () const
  
  return the final solution from selectedIterator (variables)

- **const Response & response_results** () const
  
  return the final solution from selectedIterator (response)

Private Attributes

- **Model userDefinedModel**
  
  the model to be iterated

- **Iterator selectedIterator**
  
  the iterator

8.133.1 Detailed Description

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
variables into responses.

Inheritance diagram for SingleModel:

```
+ Model
  +-- SingleModel
```

Public Member Functions

- SingleModel (ProblemDescDB &problem_db)
  constructor

- ~SingleModel ()
  destructor

Protected Member Functions

- Interface & interface ()
  return userDefinedInterface

- void derived_compute_response (const ActiveSet &set)
  (invokes a synchronous map() on userDefinedInterface)

- void derived_asynch_compute_response (const ActiveSet &set)
  (invokes an asynchronous map() on userDefinedInterface)

- const IntResponseMap & derived_synchronize ()
  (invokes synch() on userDefinedInterface)

- const IntResponseMap & derived_synchronize_nowait ()
  (invokes synch_nowait() on userDefinedInterface)

- void component_parallel_mode (short mode)
  so this virtual function redefinition is simply a sanity check.

- String local_eval_synchronization ()
  return userDefinedInterface synchronization setting

- int local_evalConcurrency ()
return userDefinedInterface asynchronous evaluation concurrency

- bool derived_master_overload () const
  evaluation (request forwarded to userDefinedInterface)

- void derived_init_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  userDefinedInterface)

- void derived_init_serial ()
  userDefinedInterface).

- void derived_set_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  (request forwarded to userDefinedInterface)

- void derived_free_communicators (const int &max_iterator_concurrency, bool recurse_flag=true)
  (request forwarded to userDefinedInterface)

- void serve ()
  Completes when a termination message is received from stop_servers().

- void stop_servers ()
  operations when SingleModel iteration is complete.

- const String & interface_id () const
  return the userDefinedInterface identifier

- int evaluation_id () const
  (request forwarded to userDefinedInterface)

- void set_evaluation_reference ()
  (request forwarded to userDefinedInterface)

- void fine_grained_evaluation_counters ()
  request fine-grained evaluation reporting within the userDefinedInterface

- void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  (request forwarded to userDefinedInterface)

Private Attributes

- Interface userDefinedInterface
  the interface used for mapping variables to responses
variables into responses.

The SingleModel class is the simplest of the derived model classes. It provides the capabilities of the original Model class, prior to the development of surrogate and nested model extensions. The derived response computation and synchronization functions utilize a single interface to perform the function evaluations.

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

- SingleModel.H
- SingleModel.C
8.135 SNLLBase Class Reference

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

Inheritance diagram for SNLLBase:

```
SNLLBase

SNLLLeastSq  SNLOptimizer
```

Public Member Functions

- **SNLLBase ()**
  - *default constructor*

- **SNLLBase (Model &model)**
  - *standard constructor*

- **~SNLLBase ()**
  - *destructor*

Protected Member Functions

- **void copy_con_vals** (const RealVector &local_fn_vals, NEWMAT::ColumnVector &g, const size_t &offset)
  - *constraint evaluator functions*

- **void copy_con_vals** (const NEWMAT::ColumnVector &g, RealVector &local_fn_vals, const size_t &offset)
  - *final solution logging*

- **void copy_con_grad** (const RealMatrix &local_fn_grads, NEWMAT::Matrix &grad_g, const size_t &offset)
  - *used by constraint evaluator functions*

- **void copy_con_hess** (const RealSymMatrixArray &local_fn_hessians, OPTPP::OptppArray<NEWMAT::SymmetricMatrix>& hess_g, const size_t &offset)
  - *used by constraint evaluator functions*

- **void snll_pre_instantiate** (const String &merit_fn, bool bound_constr_flag, const int &num_constr)
  - *method instantiation*
• void **snll_post_instantiate** (const int &num_cv, bool vendor_num_grad_flag, const String &finite_diff_type, const Real &fdss, const int &max_iter, const int &max_fn_evals, const Real &conv_tol, const Real &grad_tol, const Real &max_step, bool bound_constr_flag, const int &num_constr, short output_lev, OPTPP::OptimizeClass *the_optimizer, OPTPP::NLP0 *nlf_objective, OPTPP::FDNLFI *fd_nlf1, OPTPP::FDNLFI *fd_nlf1_con)

  *method instantiation*

• void **snll_initialize_run** (OPTPP::NLP0 *nlf_objective, OPTPP::NLP *nlp_constraint, const RealVector &init_pt, bool bound_constr_flag, const RealVector &lower_bnds, const RealVector &upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_targets)

  *method invocation*

• void **snll_post_run** (OPTPP::NLP0 *nlf_objective)

  *method instantiations*

**Static Protected Member Functions**

• static void **init_fn** (int n, NEWMAT::ColumnVector &x)

  *An initialization mechanism provided by OPT++ (not currently used).*

**Protected Attributes**

• String **searchMethod**

  *trust_region, or tr_pds*

• OPTPP::SearchStrategy **searchStrat**

  *enum: LineSearch, TrustRegion, or TrustPDS*

• OPTPP::MeritFcn **meritFn**

  *enum: NormFmu, ArgaezTapia, or VanShanno*

• bool **constantASVFlag**

  *this into mode override, reliance on duplicate detection can be avoided.*

**Static Protected Attributes**

• static **Minimizer** * optLSqInstance

  *evaluator functions in order to avoid the need for static data*

• static bool **modeOverrideFlag**
Hessian requests).

- static EvalType lastFnEvalLocn
  evaluator was the last location of a function evaluation
- static int lastEvalMode
  copy of mode from constraint evaluators
- static RealVector lastEvalVars
  copy of variables from constraint evaluators

### 8.135.1 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
8.136 SNLLLeastSq Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLLeastSq::

```
  Iterator
   |
   v
Minimizer
   |
   v
LeastSq  SNLLBase
   |
SNLLLeastSq
```

Public Member Functions

- **SNLLLeastSq (Model &model)**
  - *standard constructor*

- **SNLLLeastSq (const String &method_name, Model &model)**
  - *alternate constructor for instantiations without ProblemDescDB support*

- **~SNLLLeastSq ()**
  - *destructor*

- **void minimize_residuals ()**
  - *Performs the iterations to determine the least squares solution.*

Protected Member Functions

- **void derived_initialize_run ()**
  - *SNLLBase::snll_initialize_run(), and performs other set-up.*

- **void derived_post_run ()**
  - *and performs other solution processing*

- **void derived_finalize_run ()**
  - *restores instances and invokes LeastSq::derived_finalize_run()*
Static Private Member Functions

- **static void nlf2_evaluator_gn** (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, NEWMAT::SymmetricMatrix &hess_f, int &result_mode)
  value, gradient, and Hessian using the Gauss-Newton approximation.

- **static void constraint1_evaluator_gn** (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, int &result_mode)
  values and gradients to OPT++ Gauss-Newton methods.

- **static void constraint2_evaluator_gn** (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, OPTPP::OptppArray<NEWMAT::SymmetricMatrix> &hess_g, int &result_mode)
  values, gradients, and Hessians to OPT++ Gauss-Newton methods.

Private Attributes

- **SNLLLeastSq * prevSnllLSqInstance**
  restoration in the case of iterator/model recursion

- **OPTPP::NLP0 * nlfObjective**
  objective NLF base class pointer

- **OPTPP::NLP0 * nlfConstraint**
  constraint NLF base class pointer

- **OPTPP::NLP * nlpConstraint**
  constraint NLP pointer

- **OPTPP::NLF2 * nlf2**
  pointer to objective NLF for full Newton optimizers

- **OPTPP::NLF2 * nlf2Con**
  pointer to constraint NLF for full Newton optimizers

- **OPTPP::NLF1 * nlf1Con**
  pointer to constraint NLF for Quasi Newton optimizers

- **OPTPP::OptimizeClass * theOptimizer**
  optimizer base class pointer

- **OPTPP::OptNewton * optnewton**
  Newton optimizer pointer.

- **OPTPP::OptBCNewton * optbcnewton**
8.136 Class Reference

Bound constrained Newton optimizer ptr.

- OPTPP::OptDHNIPS * optdhnips
  Disaggregated Hessian NIPS optimizer ptr.

Static Private Attributes

- static SNLLLeastSq * snllLsqInstance
  functions in order to avoid the need for static data

8.136.1 Detailed Description

Wrapper class for the OPT++ optimization library.

The SNLLLeastSq class provides a wrapper for OPT++, a C++ optimization library of nonlinear programming and pattern search techniques from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function, a static member, or accessed by static pointer.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, max_step, gradient_tolerance, search_method, and search_scheme_size are set using OPT++’s setMaxIter(), setMaxFeval(), setFcnTol(), setMaxStep(), setGradTol(), setSearchStrategy(), and setSSS() member functions, respectively; output verbosity is used to toggle OPT++’s debug mode using the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOTA search_method specification supports 4 (value_based_line_search, gradient_based_line_search, trust_region, or tr_pds). The difference stems from the "is_expensive" flag in OPT++. If the search strategy is LineSearch and "is_expensive" is turned on, then the value_based_line_search is used. Otherwise (the "is_expensive" default is off), the algorithm will use the gradient_based_line_search. Refer to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/methods/OPTPP directory for information on OPT++ class member functions.

8.136.2 Member Function Documentation

8.136.2.1 void nlf2_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, NEWMAT::ColumnVector & grad_f, NEWMAT::SymmetricMatrix & hess_f, int & result_mode) [static, private]

value, gradient, and Hessian using the Gauss-Newton approximation.

This nlf2 evaluator function is used for the Gauss-Newton method in order to exploit the special structure of the nonlinear least squares problem. Here, fx = sum (T_i - Tbar_i)^2 and Response is made up of residual functions and their derivatives (which are returned from the Response object).
8.136.2.2  void constraint1_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector & x,
             NEWMAT::ColumnVector & g, NEWMAT::Matrix & grad_g, int & result_mode)  [static, private]

values and gradients to OPT++ Gauss-Newton methods.
While it does not employ the Gauss-Newton approximation, it is distinct from constraint1_evaluator() due to its
need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with
diaggregated Hessian NIPS and is currently active.

8.136.2.3  static void constraint2_evaluator_gn (int mode, int n, const NEWMAT::ColumnVector & x,
             NEWMAT::ColumnVector & g, NEWMAT::Matrix & grad_g, OPTPP::OptppArray<
             NEWMAT::SymmetricMatrix > & hess_g, int & result_mode)  [static, private]

values, gradients, and Hessians to OPT++ Gauss-Newton methods.
While it does not employ the Gauss-Newton approximation, it is distinct from constraint2_evaluator() due to its
need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with
full Newton NIPS and is currently inactive.

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

- SNLLLeastSq.H
- SNLLLeastSq.C
8.137 SNLLOptimizer Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLOptimizer::

```
   SNLLOptimizer
     ├── Iterator
     │    ├── Minimizer
     │    │    └── Optimizer
     │    └── SNLLBase
```

Public Member Functions

- **SNLLOptimizer (Model &model)**
  *standard constructor*

- **SNLLOptimizer (const String &method_name, Model &model)**
  *alternate constructor for instantiations "on the fly"*

- **SNLLOptimizer (const RealVector &initial_pt, const RealVector &var_l_bnds, const RealVector &var_u_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_tgts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, void(*user_obj_eval)(int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode), void(*user_con_eval)(int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode))
  *alternate constructor for instantiations "on the fly"*

- ~SNLLOptimizer ()
  *destructor*

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

Protected Member Functions

- void **derived_initialize_run ()**
  *SNLLBase::snll_initialize_run(), and performs other set-up.*
• void `derived_post_run()`
  <i>performs data recovery and calls Optimizer::derived_post_run()</i>

• void `derived_finalize_run()`
  <i>performs cleanup, restores instances and calls parent finalize</i>

**Static Private Member Functions**

• static void `nlf0_evaluator` (int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, int &result_mode)
  <i>require only function values.</i>

• static void `nlf1_evaluator` (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode)
  <i>values and gradients to OPT++ methods.</i>

• static void `nlf2_evaluator` (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, NEWMAT::SymmetricMatrix &hess_f, int &result_mode)
  <i>values, gradients, and Hessians to OPT++ methods.</i>

• static void `constraint0_evaluator` (int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, int &result_mode)
  <i>only constraint values.</i>

• static void `constraint1_evaluator` (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, int &result_mode)
  <i>values and gradients to OPT++ methods.</i>

• static void `constraint2_evaluator` (int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::ColumnVector &g, NEWMAT::Matrix &grad_g, OPTPP::OptppArray<NEWMAT::SymmetricMatrix> &hess_g, int &result_mode)
  <i>values, gradients, and Hessians to OPT++ methods.</i>

**Private Attributes**

• SNLLOptimizer * `prevSnllOptInstance`
  <i>restoration in the case of iterator/model recursion</i>

• OPTPP::NLP0 * `nlfObjective`
  <i>objective NLF base class pointer</i>

• OPTPP::NLP0 * `nlfConstraint`
  <i>constraint NLF base class pointer</i>
8.137 SNLLOptimizer Class Reference

- **OPTPP::NLP** * nlpConstraint
  
  constraint NLP pointer

- **OPTPP::NLF0** * nlf0
  
  pointer to objective NLF for nongradient optimizers

- **OPTPP::NLF1** * nlf1
  
  pointer to objective NLF for (analytic) gradient-based optimizers

- **OPTPP::NLF1** * nlf1Con
  
  pointer to constraint NLF for (analytic) gradient-based optimizers

- **OPTPP::FDNLF1** * fdnlf1
  
  pointer to objective NLF for (finite diff) gradient-based optimizers

- **OPTPP::FDNLF1** * fdnlf1Con
  
  pointer to constraint NLF for (finite diff) gradient-based optimizers

- **OPTPP::NLF2** * nlf2
  
  pointer to objective NLF for full Newton optimizers

- **OPTPP::NLF2** * nlf2Con
  
  pointer to constraint NLF for full Newton optimizers

- **OPTPP::OptimizeClass** * theOptimizer
  
  optimizer base class pointer

- **OPTPP::OptPDS** * optpds
  
  PDS optimizer pointer.

- **OPTPP::OptCG** * optcg
  
  CG optimizer pointer.

- **OPTPP::OptLBFGS** * optlbfgs
  
  L-BFGS optimizer pointer.

- **OPTPP::OptNewton** * optnewton
  
  Newton optimizer pointer.

- **OPTPP::OptQNewton** * optqnewton
  
  Quasi-Newton optimizer pointer.

- **OPTPP::OptFDNewton** * optfdnewton
  
  Finite Difference Newton opt pointer.

- **OPTPP::OptBCNewton** * optbcnewton
Bound constrained Newton opt pointer.

- OPTPP::OptBCQNewton * optbcqnewton
  Bnd constrained Quasi-Newton opt ptr.

- OPTPP::OptBCFDNewton * optbcfdnewton
  Bnd constrained FD-Newton opt ptr.

- OPTPP::OptNIPS * optnips
  NIPS optimizer pointer.

- OPTPP::OptQNIPS * optqnips
  Quasi-Newton NIPS optimizer pointer.

- OPTPP::OptFDNIPS * optfdnips
  Finite Difference NIPS opt pointer.

- String setUpType
  NonDReliability currently uses the user_functions mode.

- RealVector initialPoint
  holds initial point passed in for "user_functions" mode.

- RealVector lowerBounds
  holds variable lower bounds passed in for "user_functions" mode.

- RealVector upperBounds
  holds variable upper bounds passed in for "user_functions" mode.

Static Private Attributes

- static SNLLOptimizer * snloptInstance
  functions in order to avoid the need for static data

8.137.1 Detailed Description

Wrapper class for the OPT++ optimization library.

The SNLLOptimizer class provides a wrapper for OPT++, a C++ optimization library of nonlinear programming and pattern search techniques from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function, a static member, or accessed by static pointer.
The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, max_step, gradient_tolerance, search_method, and search_scheme_size are set using OPT++’s setMaxIter(), setMaxFeval(), setFcnTol(), setMaxStep(), setGradTol(), setSearchStrategy(), and setSSS() member functions, respectively; output verbosity is used to toggle OPT++’s debug mode using the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOTA search_method specification supports 4 (value_based_line_search, gradient_based_line_search, trust_region, or tr_pds). The difference stems from the "is_expensive" flag in OPT++. If the search strategy is LineSearch and "is_expensive" is turned on, then the value_based_line_search is used. Otherwise (the "is_expensive" default is off), the algorithm will use the gradient_based_line_search. Refer to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/methods/OPTPP directory for information on OPT++ class member functions.

8.137.2 Constructor & Destructor Documentation

8.137.2.1 SNLLOptimizer (Model & model)

standard constructor

This constructor is used for normal instantiations using data from the ProblemDescDB.

8.137.2.2 SNLLOptimizer (const String & method_name, Model & model)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

8.137.2.3 SNLLOptimizer (const RealVector & initial_pt, const RealVector & var_l_bnds, const RealVector & var_u_bnds, const RealMatrix & lin_ineq_coeffs, const RealVector & lin_ineq_l_bnds, const RealVector & lin_ineq_u_bnds, const RealMatrix & lin_eq_coeffs, const RealVector & lin_eq_tgts, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts, void(*)(int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Real &f, NEWMAT::ColumnVector &grad_f, int &result_mode) user_obj_eval, void(*)(int mode, int n, const NEWMAT::ColumnVector &x, NEWMAT::Matrix &grad_g, int &result_mode) user_con_eval)

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers.

8.137.3 Member Function Documentation

8.137.3.1 void nlf0_evaluator (int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, int & result_mode) [static, private]

require only function values.
For use when DAKOTA computes f and gradients are not directly available. This is used by nongradient-based optimizers such as PDS and by gradient-based optimizers in vendor numerical gradient mode (opt++’s internal finite difference routine is used).

8.137.3.2  void nlf1_evaluator (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, NEWMAT::ColumnVector & grad_f, int & result_mode)  [static, private]

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.

8.137.3.3  void nlf2_evaluator (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::Real & f, NEWMAT::ColumnVector & grad_f, NEWMAT::SymmetricMatrix & hess_f, int & result_mode)  [static, private]

values, gradients, and Hessians to OPT++ methods.
For use when DAKOTA receives f, df/dX, & d²f/dx² 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.

8.137.3.4  void constraint0_evaluator (int n, const NEWMAT::ColumnVector & x, NEWMAT::ColumnVector & g, int & result_mode)  [static, private]

only constraint values.
For use when DAKOTA computes g and gradients are not directly available. This is used by nongradient-based optimizers and by gradient-based optimizers in vendor numerical gradient mode (opt++’s internal finite difference routine is used).

8.137.3.5  void constraint1_evaluator (int mode, int n, const NEWMAT::ColumnVector & x, NEWMAT::ColumnVector & g, NEWMAT::Matrix & grad_g, int & result_mode)  [static, private]

values and gradients to OPT++ methods.
For use when DAKOTA computes g and dg/dX (regardless of gradientType). Vendor numerical gradient case is handled by constraint0_evaluator.

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

- SNLLOptimizer.H
- SNLLOptimizer.C


8.138 SOLBase Class Reference

Base class for Stanford SOL software.

Inheritance diagram for SOLBase::

```
SOLBase
   \______________  
     |               |
  NLSSOLLeastSq    NPSOLOptimizer
```

Public Member Functions

- **SOLBase ()**
  default constructor

- **SOLBase (Model &model)**
  standard constructor

- **~SOLBase ()**
  destructor

Protected Member Functions

- **void allocate_arrays (const int &num_cv, const size_t &num_nln_con, const RealMatrix &lin_ineq_coeffs, const RealMatrix &lin_eq_coeffs)**
  Allocates miscellaneous arrays for the SOL algorithms.

- **void deallocate_arrays ()**
  Deallocates memory previously allocated by allocate_arrays().

- **void allocate_workspace (const int &num_cv, const int &num_nln_con, const int &num_lin_con, const int &num_lsq)**
  Allocates real and integer workspaces for the SOL algorithms.

- **void set_options (bool speculative_flag, bool vendor_num_grad_flag, short output_lev, const int &verify_lev, const Real &fn_prec, const Real &linesrch_tol, const int &max_iter, const Real &constr_tol, const Real &conv_tol, const String &grad_type, const Real &fdss)**
  Sets SOL method options using calls to npoptn2.

- **void augment_bounds (RealVector &augmented_l_bnds, RealVector &augmented_u_bnds, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealVector &lin_eq_targets, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_targets)**
  Augments variable bounds with linear and nonlinear constraint bounds.
Static Protected Member Functions

- static void constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int *needc, double *x, double *c, double *cjac, int &nstate)
  
  derivatives of the nonlinear constraint functions

Protected Attributes

- int realWorkspaceSize
  size of realWorkSpace

- int intWorkspaceSize
  size of intWorkSpace

- RealArray realWorkspace
  real work space for NPSOL/NLSSOL

- IntArray intWorkspace
  int work space for NPSOL/NLSSOL

- int nlnConstraintArraySize
  used for non-zero array sizing (nonlinear constraints)

- int linConstraintArraySize
  used for non-zero array sizing (linear constraints)

- RealArray cLambda
  CLAMBDA from NPSOL manual: Lagrange multipliers.

- IntArray constraintState
  ISTATE from NPSOL manual: constraint status.

- int informResult
  INFORM from NPSOL manual: optimization status on exit.

- int numberIterations
  ITER from NPSOL manual: number of (major) iterations performed.

- int boundsArraySize
  nonlinear constraint bounds)

- double * linConstraintMatrixF77
  [A] matrix from NPSOL manual: linear constraint coefficients

- double * upperFactorHessianF77
the Lagrangian.

- double * constraintJacMatrixF77
  
  \[ CJAC \] matrix from NPSOL manual: nonlinear constraint Jacobian

- int fnEvalCnt
  
  counter for testing against maxFunctionEvals

- size_t constrOffset
  
  and NPSOLOptimizer::numObjectiveFns

### Static Protected Attributes

- static SOLBase * solInstance
  
  functions in order to avoid the need for static data

- static Minimizer * optLsqInstance
  
  evaluator functions in order to avoid the need for static data

### 8.138.1 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
8.139 Strategy Class Reference

Base class for the strategy class hierarchy.

Inheritance diagram for Strategy::

```
Strategy
├── ConcurrentStrategy
├── HybridStrategy
├── SingleMethodStrategy
└── CollaborativeHybridStrategy
    └── EmbeddedHybridStrategy
        └── SequentialHybridStrategy
```

Public Member Functions

- **Strategy ()**
  
  *default constructor*

- **Strategy (ProblemDescDB &problem_db)**
  
  *envelope constructor*

- **Strategy (const Strategy &strat)**
  
  *copy constructor*

- **virtual ~Strategy ()**
  
  *destructor*

- **Strategy operator= (const Strategy &strat)**
  
  *assignment operator*

- **virtual void run_strategy ()**
  
  *the model(s). Called from main.C.*

- **virtual const Variables & variables_results () const**
  
  *return the final strategy solution (variables)*

- **virtual const Response & response_results () const**
  
  *return the final strategy solution (response)*

- **ProblemDescDB & problem_description_db () const**
  
  *returns the problem description database (probDescDB)*
Protected Member Functions

- **Strategy** *(BaseConstructor, ProblemDescDB &problem_db)*
  
  derived class constructors - Coplien, p. 139

- virtual void **initialize_iterator** (int index)
  
  scheduling function (serve_iterators() or static_schedule_iterators())

- virtual void **pack_parameters_buffer** (MPIPackBuffer &send_buffer, int job_index)
  
  pack a send_buffer for assigning an iterator job to a server

- virtual void **unpack_parameters_buffer** (MPIUnpackBuffer &recv_buffer)
  
  unpack a recv_buffer for accepting an iterator job from the scheduler

- virtual void **pack_results_buffer** (MPIPackBuffer &send_buffer, int job_index)
  
  pack a send_buffer for returning iterator results from a server

- virtual void **unpack_results_buffer** (MPIUnpackBuffer &recv_buffer, int job_index)
  
  unpack a recv_buffer for accepting iterator results from a server

- virtual void **update_local_results** (int job_index)
  
  update local prpResults with current iteration results

- void **init_iterator_parallelism** ()
  
  parallel configuration attributes, and managing outputs and restart.

- void **init_iterator** (Iterator &the_iterator, Model &the_model)
  
  convenience function for allocating comms prior to running an iterator

- void **run_iterator** (Iterator &the_iterator, Model &the_model)
  
  due to use by MINLPNode.

- void **free_iterator** (Iterator &the_iterator, Model &the_model)
  
  convenience function for deallocating comms after running an iterator

- void **schedule_iterators** (Iterator &the_iterator, Model &the_model)
  
  static_schedule_iterators()

- void **self_schedule_iterators** (Model &the_model)
  
  among slave iterator servers (called by derived run_strategy())

- void **serve_iterators** (Iterator &the_iterator, Model &the_model)
  
  assigned by the strategy master (called by derived run_strategy())

- void **static_schedule_iterators** (Iterator &the_iterator, Model &the_model)
  
  (called by derived run_strategy())
Protected Attributes

- **ProblemDescDB & probDescDB**
  class member reference to the problem description database

- **ParallelLibrary & parallelLib**
  class member reference to the parallel library

- **String strategyName**
  type of strategy: single_method, hybrid, multi_start, or pareto_set.

- **bool stratIterMessagePass**
  flag for message passing at si level

- **bool stratIterDedMaster**
  flag for dedicated master part. at si level

- **int worldRank**
  processor rank in MPI_COMM_WORLD

- **int worldSize**
  size of MPI_COMM_WORLD

- **int iteratorCommRank**
  processor rank in iteratorComm

- **int iteratorCommSize**
  number of processors in iteratorComm

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

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

- **bool graph2DFlag**
  flag for using 2D graphics plots

- **bool tabularDataFlag**
  flag for file tabulation of graphics data

- **String tabularDataFile**
  filename for tabulation of graphics data

- **int maxConcurrency**
  maximum iterator concurrency possible in Strategy
8.139 Strategy Class Reference

- **int numIteratorJobs**
  
  *number of iterator executions to schedule*

- **PRPArray prpResults**
  
  *array of results corresponding to numIteratorJobs*

- **int paramsMsgLen**
  
  *length of MPI buffer for parameterSets instance(s)*

- **int resultsMsgLen**
  
  *length of MPI buffer for prpResults instance(s)*

**Private Member Functions**

- **Strategy * get_strategy ()**
  
  *Used by the envelope to instantiate the correct letter class.*

**Private Attributes**

- **Strategy * strategyRep**
  
  *pointer to the letter (initialized only for the envelope)*

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

### 8.139.1 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.

### 8.139.2 Constructor & Destructor Documentation

**8.139.2.1 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.

### 8.139.2.2 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.

### 8.139.2.3 Strategy (const Strategy & strat)

**copy constructor**

Copy constructor manages sharing of strategyRep and incrementing of referenceCount.

### 8.139.2.4 ~Strategy () [virtual]

**destructor**

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

### 8.139.2.5 Strategy (BaseConstructor, ProblemDescDB & problem_db) [protected]

**derived class constructors - Coplien, p. 139)**

This constructor is the one which must build the base class data for all inherited strategies. get_strategy() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_strategy() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Strategy).

### 8.139.3 Member Function Documentation

#### 8.139.3.1 Strategy operator= (const Strategy & strat)

**assignment operator**


#### 8.139.3.2 void pack_parameters_buffer (MPIPackBuffer & send_buffer, int job_index) [protected, virtual]

pack a send_buffer for assigning an iterator job to a server
This virtual function redefinition is executed on the dedicated master processor for self scheduling. It is not used for peer partitions.
Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

8.139.3.3 void unpack_parameters_buffer (MPIUnpackBuffer & recv_buffer) [protected, virtual]
unpack a recv_buffer for accepting an iterator job from the scheduler
This virtual function redefinition is executed on an iterator server for dedicated master self scheduling. It is not used for peer partitions.
Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

8.139.3.4 void pack_results_buffer (MPIPackBuffer & send_buffer, int job_index) [protected, virtual]
pack a send_buffer for returning iterator results from a server
This virtual function redefinition is executed either on an iterator server for dedicated master self scheduling or on peers 2 through n for static scheduling.
Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

8.139.3.5 void unpack_results_buffer (MPIUnpackBuffer & recv_buffer, int job_index) [protected, virtual]
unpack a recv_buffer for accepting iterator results from a server
This virtual function redefinition is executed on a strategy master (either the dedicated master processor for self scheduling or peer 1 for static scheduling).
Reimplemented in ConcurrentStrategy, and SequentialHybridStrategy.

8.139.3.6 void init_iterator_parallelism () [protected]
parallel configuration attributes, and managing outputs and restart.
This function is called from derived class constructors once maxConcurrency is defined but prior to instantiateing Iterators and Models.

8.139.3.7 void init_iterator (Iterator & the_iterator, Model & the_model) [protected]
convenience function for allocating comms prior to running an iterator
This is a convenience function for encapsulating the allocation of communicators prior to running an iterator. It does not require a strategyRep forward since it is only used by letter objects.
8.139.3.8 **void run_iterator (Iterator & the_iterator, Model & the_model)**  [protected]

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.

8.139.3.9 **void free_iterator (Iterator & the_iterator, Model & the_model)**  [protected]

convenience function for deallocating comms after running an iterator
This is a convenience function for encapsulating the deallocation of communicators after running an iterator. It does not require a strategyRep forward since it is only used by letter objects.

8.139.3.10 **void schedule_iterators (Iterator & the_iterator, Model & the_model)**  [protected]

static_schedule_iterators()
This implementation supports the scheduling of multiple jobs using a single iterator/model pair. Additional future (overloaded) implementations could involve independent iterator instances.

8.139.3.11 **void self_schedule_iterators (Model & the_model)**  [protected]

among slave iterator servers (called by derived run_strategy())
This function is adapted from ApplicationInterface::self_schedule_evaluations().

8.139.3.12 **void serve_iterators (Iterator & the_iterator, Model & the_model)**  [protected]

assigned by the strategy master (called by derived run_strategy())
This function is similar in structure to ApplicationInterface::serve_evaluations_synch().

8.139.3.13 **Strategy * get_strategy ()**  [private]

Used by the envelope to instantiate the correct letter class.
Used only by the envelope constructor to initialize strategyRep to the appropriate derived type, as given by the strategyName attribute.
The documentation for this class was generated from the following files:

- DakotaStrategy.H
- DakotaStrategy.C
8.140 String Class Reference

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

Public Member Functions

- **String ()**
  Default constructor.

- **String (const String &a)**
  Copy constructor for incoming String.

- **String (const String &a, size_t start_index, size_t num_items)**
  Copy constructor for portion of incoming String.

- **String (const char ∗c_string)**
  Copy constructor for incoming char* array.

- **String (const std::string &a)**
  Copy constructor for incoming base string.

- **~String ()**
  Destructor.

- **String & operator= (const String &)**
  Assignment operator for incoming String.

- **String & operator= (const std::string &)**
  Assignment operator for incoming base string.

- **String & operator= (const char ∗c_string)**
  Assignment operator for incoming char* array.

- **operator const char ∗ () const**
  The operator() returns pointer to standard C char array.

- **String & toUpper ()**
  Convert to upper case string.

- **void upper ()**

- **String & toLower ()**
  Convert to lower case string.

- **void lower ()**

- **bool contains (const char ∗sub_string) const**
8.140.1 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 using the std::string interface or the (legacy) RogueWave RWCString API.

8.140.2 Member Function Documentation

8.140.2.1 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.

8.140.2.2 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.

8.140.2.3 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.

8.140.2.4 bool contains (const char * sub_string) const [inline]

Returns true if String contains char* substring.

Returns true if the String contains the char* sub_string. Uses the STL find() method.

Returns true if String starts with char* substring.

Returns true if String ends with char* substring.
8.140.2.5 bool begins (const char * sub_string) const [inline]

Returns true if String starts with char* substring.
Returns true if the String begins with the char* sub_string. Uses the STL compare() method.

8.140.2.6 bool ends (const char * sub_string) const [inline]

Returns true if String ends with char* substring.
Returns true if the String ends with the char* sub_string. Uses the STL compare() method.

8.140.2.7 char * data () const [inline]

Returns pointer to standard C char array.
Returns a pointer to C style char array. Needed to mimic the Rogue Wave string class. USE WITH CARE.
The documentation for this class was generated from the following files:

- DakotaString.H
- DakotaString.C
8.141 SurfpackApproximation Class Reference

Interface between Surfpack and Dakota.

Inheritance diagram for SurfpackApproximation:

```
          Approximation
            |   |
            v   v
SurfpackApproximation
```

Public Member Functions

- `SurfpackApproximation()`
  *default constructor*

- `SurfpackApproximation(const ProblemDescDB &problem_db, const size_t &num_acv)`
  *standard constructor: Surfpack surface of appropriate type will be created*

- `~SurfpackApproximation()`
  *destructor*

Protected Member Functions

- `int min_coefficients() const`
  *build the derived class approximation type in numVars dimensions*

- `int recommended_coefficients() const`
  *build the derived class approximation type in numVars dimensions*

- `void find_coefficients()`
  *and the appropriate Surfpack build method will be invoked*

- `const Real & get_value(const RealVector &x)`
  *Return the value of the Surfpack surface for a given parameter vector x.*

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

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

- `const Real & get_diagnostic(const String &metric_type)`
retrieve the diagnostic metric for the diagnostic type specified

- const bool diagnostics_available ()
  check if the diagnostics are available (true for the Surfpack types)

**Private Member Functions**

- void checkForEqualityConstraints ()
  point, gradient, and/or hessian

- SurfData * surrogates_to_surf_data ()
  copy from SurrogateDataPoint to SurfPoint/SurfData

**Private Attributes**

- SurfpackModel * model
  The native Surfpack approximation.

- SurfpackModelFactory * factory
  factory for the SurfpackModel instance

- SurfData * surfData
  The data used to build the approximation, in Surfpack format.

### 8.141.1 Detailed Description

Interface between Surfpack and Dakota.

The SurfpackApproximation class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SurfpackApproximation builds a Surfpack Surface object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial neural networks, radial basis function network, or multivariate adaptive regression splines (MARS).

### 8.141.2 Member Function Documentation

#### 8.141.2.1 void find_coefficients () [protected, virtual]

and the appropriate Surfpack build method will be invoked

surfData will be deleted in dtor

**Todo**

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

8.141.2.2 const RealSymMatrix & get_hessian (const RealVector & x)  [protected, virtual]
retrieve the approximate function Hessian for a given parameter vector x

Todo
Make this acceptably efficient

Reimplemented from Approximation.

8.141.2.3 void checkForEqualityConstraints ()  [private]
point, gradient, and/or hessian
If there is an anchor point, add an equality constraint for its response value. Also add constraints for gradient and hessian, if applicable.

Todo
improve efficiency of conversion

8.141.2.4 SurfData * surrogates_to_surf_data ()  [private]
copy from SurrogateDataPoint to SurfPoint/SurfData
Copy the data stored in Dakota-style SurrogateDataPoint objects into Surfpack-style SurfPoint and SurfData objects.
The documentation for this class was generated from the following files:

• SurfpackApproximation.H
• SurfpackApproximation.C
8.142 SurrBasedGlobalMinimizer Class Reference

and updates a global surrogate model without trust region controls

Inheritance diagram for SurrBasedGlobalMinimizer::

```
       Iterator
          |
          Minimizer
            |
            SurrBasedMinimizer
              |
              SurrBasedGlobalMinimizer
```

Public Member Functions

- `SurrBasedGlobalMinimizer(Model &model)`
  - constructor

- `~SurrBasedGlobalMinimizer()`
  - destructor

Protected Member Functions

- `bool returns_multiple_points() const`
  - Global surrogate-based methods can return multiple points.

Private Member Functions

- `void minimize_surrogates()`
  - optimizing on and improving surrogates of the response functions.

Private Attributes

- `bool replacePoints`
  - than continuing to append, during construction of the next surrogate
8.142.1 Detailed Description

and updates a global surrogate model without trust region controls

This method uses a SurrogateModel to perform minimization (optimization or nonlinear least squares) through a set of iterations. At each iteration, a surrogate is built, the surrogate is minimized, and the optimal points from the surrogate are then evaluated with the "true" function, to generate new points upon which the surrogate for the next iteration is built.

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

- SurrBasedGlobalMinimizer.H
- SurrBasedGlobalMinimizer.C
8.143 SurrBasedLocalMinimizer Class Reference

and nonlinear least squares.

Inheritance diagram for SurrBasedLocalMinimizer::

```
  SurrBasedMinimizer
  |                  |
  |                  |
SurrBasedLocalMinimizer
```

Public Member Functions

- `SurrBasedLocalMinimizer (Model &model)`
  
  constructor

- `~SurrBasedLocalMinimizer ()`
  
  destructor

Private Member Functions

- void `minimize_surrogates ()`
  
  global, or hierarchical surrogates over a series of trust regions.

- void `reset ()`

  reset convergence controls in case of multiple SBLM executions

- bool `tr_bounds (const RealVector &global_lower_bnds, const RealVector &global_upper_bnds, RealVector &tr_lower_bnds, RealVector &tr_upper_bnds)`

  compute current trust region bounds

- void `find_center_truth (const Iterator &dace_iterator, Model &truth_model)`

  retrieve responseCenterTruth if possible, evaluate it if not

- void `find_center_approx ()`

  retrieve responseCenter_approx if possible, evaluate it if not

- void `hard_convergence_check (const Response &response_truth, const RealVector &c_vars, const RealVector &lower_bnds, const RealVector &upper_bnds)`
merit function near zero)

- void tr_ratio_check (const RealVector &c_vars_star, const RealVector &tr_lower_bounds, const RealVector &tr_upper_bounds)
  
  region resizing and check for soft convergence (diminishing returns)

- void update_penalty (const RealVector &fns_center_truth, const RealVector &fns_star_truth)
  
  initialize and update the penaltyParameter

- void relax_constraints (const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  relax constraints by updating bounds when current iterate is infeasible

Static Private Member Functions

- static void approx_subprob_objective_eval (const Variables &surrogate_vars, const Variables &recast_vars, const Response &surrogate_response, Response &recast_response)
  
  static function used to define the approximate subproblem objective.

- static void approx_subprob_constraint_eval (const Variables &surrogate_vars, const Variables &recast_vars, const Response &surrogate_response, Response &recast_response)
  
  static function used to define the approximate subproblem constraints.

- static void hom_objective_eval (int &mode, int &n, double *tau_and_x, double &f, double *grad_f, int &)
  
  homotopy constraint relaxation formulation.

- static void hom_constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int *needc, double *tau_and_, double *c, double *cjac, int &nstate)
  
  homotopy constraint relaxation formulation.

Private Attributes

- Real origTrustRegionFactor
  
  original user specification for trustRegionFactor

- Real trustRegionFactor
  
  bound - lower bound for each design variable).

- Real minTrustRegionFactor
  
  factor is reduced below the value of minTrustRegionFactor

- Real trRatioContractValue
  
  trust region ratio min value: contract tr if ratio below this value
8.143 SurrBasedLocalMinimizer Class Reference

- Real $trRatioExpandValue$
  trust region ratio sufficient value: expand $tr$ if ratio above this value

- Real $gammaContract$
  trust region contraction factor

- Real $gammaExpand$
  trust region expansion factor

- short $approxSubProbObj$
  or AUGMENTED_LAGRANGIAN_OBJ

- short $approxSubProbCon$
  ORIGINAL_CON.

- Model $approxSubProbModel$
  involve a RecastModel recursion applied to iteratedModel

- bool $recastSubProb$
  flag to indicate when approxSubProbModel involves a RecastModel recursion

- short $trConstraintRelax$
  points: NO_RELAX or HOMOTOPY

- short $meritFnType$
  ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, or AUGMENTED_LAGRANGIAN_MERIT.

- short $acceptLogic$
  type of iterate acceptance test logic: FILTER or TR_RATIO

- int $penaltyIterOffset$
  for adaptive_penalty merit functions

- short $convergenceFlag$
  code indicating satisfaction of hard or soft convergence conditions

- short $softConvCount$
  count reaches softConvLimit, stop SBLM.

- short $softConvLimit$
  exceeded by softConvCount, stop SBLM.

- bool $truthGradientFlag$
  flags the use/availability of truth gradients within the SBLM process

- bool $approxGradientFlag$
flags the use/availability of surrogate gradients within the SBLM process

- bool truthHessianFlag
  flags the use/availability of truth Hessians within the SBLM process

- bool approxHessianFlag
  flags the use/availability of surrogate Hessians within the SBLM process

- bool correctionFlag
  of each trust region

- bool globalApproxFlag
  flags the use of a global data fit surrogate (rsm, ann, mars, kriging)

- bool multiptApproxFlag
  flags the use of a multipoint data fit surrogate (TANA)

- bool localApproxFlag
  flags the use of a local data fit surrogate (Taylor series)

- bool hierarchApproxFlag
  flags the use of a model hierarchy/multifidelity surrogate

- bool newCenterFlag
  a new trust region center

- bool daceCenterPtFlag
  evaluations for global approximations (CCD, Box-Behnken)

- bool multiLayerBypassFlag
  (responseCenterTruth and responseStarTruth).

- bool useGradsFlag
  to be evaluated for each DACE point in global surrogate builds.

- RealVector nonlinIneqLowerBndsSlack
  individual violations of nonlinear inequality constraint lower bounds

- RealVector nonlinIneqUpperBndsSlack
  individual violations of nonlinear inequality constraint upper bounds

- RealVector nonlinEqTargetsSlack
  individual violations of nonlinear equality constraint targets

- Real tau
  constraint relaxation parameter
8.143 SurrBasedLocalMinimizer Class Reference

- Real alpha
  constraint relaxation parameter backoff parameter (multiplier)

- Variables varsCenter
  variables at the trust region center

- Response responseCenterApprox
  approx response at trust region center

- Response responseStarApprox
  approx response at SBLM cycle minimum

- Response responseCenterTruth
  truth response at trust region center

- Response responseStarTruth
  truth response at SBLM cycle minimum

Static Private Attributes

- static SurrBasedLocalMinimizer * sblmInstance
  pointer to SBLM instance used in static member functions

8.143.1 Detailed Description

and nonlinear least squares.

This minimizer uses a SurrogateModel to perform minimization based on local, global, or hierarchical surrogates. It achieves provable convergence through the use of a sequence of trust regions and the application of surrogate corrections at the trust region centers.

8.143.2 Member Function Documentation

8.143.2.1 void minimize_surrogates () [private, virtual]

global, or hierarchical surrogates over a series of trust regions.

Trust region-based strategy to perform surrogate-based optimization in subregions (trust regions) of the parameter space. The minimizer operates on approximations in lieu of the more expensive simulation-based response functions. The size of the trust region is varied according to the goodness of the agreement between the approximations and the true response functions.

Implements SurrBasedMinimizer.
8.143.2.2 void hard_convergence_check (const Response & response_truth, const RealVector & c_vars, const RealVector & lower_bnds, const RealVector & upper_bnds) [private]

merit function near zero)

The hard convergence check computes the gradient of the merit function at the trust region center, performs a projection for active bound constraints (removing any gradient component directed into an active bound), and signals convergence if the 2-norm of this projected gradient is less than convergenceTol.

8.143.2.3 void tr_ratio_check (const RealVector & c_vars_star, const RealVector & tr_lower_bnds, const RealVector & tr_upper_bnds) [private]

region resizing) and check for soft convergence (diminishing returns)

Assess acceptance of SBLM iterate (trust region ratio or filter) and compute soft convergence metrics (number of consecutive failures, min trust region size, etc.) to assess whether the convergence rate has decreased to a point where the process should be terminated (diminishing returns).

8.143.2.4 void update_penalty (const RealVector & fns_center_truth, const RealVector & fns_star_truth) [private]

initialize and update the penaltyParameter

Scaling of the penalty value is important to avoid rejecting SBLM iterates which must increase the objective to achieve a reduction in constraint violation. In the basic penalty case, the penalty is ramped exponentially based on the iteration counter. In the adaptive case, the ratio of relative change between center and star points for the objective and constraint violation values is used to rescale penalty values.

8.143.2.5 void approx_subprob_objective_eval (const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response) [static, private]

static function used to define the approximate subproblem objective.

Objective functions evaluator for solution of approximate subproblem using a RecastModel.

8.143.2.6 void approx_subprob_constraint_eval (const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response) [static, private]

static function used to define the approximate subproblem constraints.

Constraint functions evaluator for solution of approximate subproblem using a RecastModel.

8.143.2.7 void hom_objective_eval (int & mode, int & n, double * tau_and_x, double & f, double * grad_f, int & ) [static, private]

homotopy constraint relaxation formulation.
NPSOL objective functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.

8.143.2.8 `void hom_constraint_eval (int & mode, int & ncnln, int & n, int & nrowj, int * needc, double * tau_and_x, double * c, double * jac, int & nstate)` [static, private]

homotopy constraint relaxation formulation.

NPSOL constraint functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.

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

- SurrBasedLocalMinimizer.H
- SurrBasedLocalMinimizer.C
8.144 SurrBasedMinimizer Class Reference

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

Inheritance diagram for SurrBasedMinimizer:

![Inheritance Diagram]

Protected Member Functions

- **SurrBasedMinimizer (Model &model)**
  *Constructor*

- **~SurrBasedMinimizer ()**
  *Destructor*

- **void initialize_graphics (bool graph_2d, bool tabular_data, const String &tabular_file)**
  *Initialize graphics customized for surrogate-based iteration*

- **void run ()**
  *And may contain pre/post steps in lieu of separate pre/post*

- **void print_results (std::ostream &s)**
- **virtual void minimize_surrogates ()=0**
  *Approach. Redefines the Iterator::run() virtual function.*

- **void update_lagrange_multipliers (const RealVector &fn_vals, const RealMatrix &fn_grads)**
  *Initialize and update Lagrange multipliers for basic Lagrangian*

- **void update_augmented_lagrange_multipliers (const RealVector &fn_vals)**
  *Initialize and update the Lagrange multipliers for augmented Lagrangian*

- **bool update_filter (const RealVector &fn_vals)**
  *Update a filter from a set of function values*

- **Real lagrangian_merit (const RealVector &fn_vals, const RealVector &lnl_ineq_l_bnds, const RealVector &lnl_ineq_u_bnds, const RealVector &lnl_eq_tgts)**
compute a Lagrangian function from a set of function values

- void lagrangian_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealVector &lag_grad)
  compute the gradient of the Lagrangian function

- Real augmented_lagrangian_merit (const RealVector &fn_vals, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts)
  compute an augmented Lagrangian function from a set of function values

- void augmented_lagrangian_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealVector &alag_grad)
  compute the gradient of the augmented Lagrangian function

- Real penalty_merit (const RealVector &fn_vals)
  compute a penalty function from a set of function values

- void penalty_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, RealVector &pen_grad)
  compute the gradient of the penalty function

- Real objective (const RealVector &fn_vals)
  compute a composite objective value from one or more objective functions

- void objective_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, RealVector &obj_grad)
  compute the gradient of the composite objective function

- Real constraint_violation (const RealVector &fn_vals, const Real &constraint_tol)
  compute the constraint violation from a set of function values

Protected Attributes

- Iterator approxSubProbMinimizer
  approximate subproblem on each surrogate-based iteration

- int sbIterNum
  surrogate-based minimization iteration number

- bool optimizationFlag
  flag for use where optimization and NLS must be distinguished

- RealVectorArray sbFilter
  constraint violation) for iterate selection/rejection
- RealVector `lagrangeMult`
  *Lagrange multipliers for basic Lagrangian calculations.*

- RealVector `augLagrangeMult`
  *Lagrange multipliers for augmented Lagrangian calculations.*

- Real `penaltyParameter`
  *penalty calculations; increased in update_penalty()

- RealVector `origNonlinIneqLowerBnds`
  *original nonlinear inequality constraint lower bounds (no relaxation)*

- RealVector `origNonlinIneqUpperBnds`
  *original nonlinear inequality constraint upper bounds (no relaxation)*

- RealVector `origNonlinEqTargets`
  *original nonlinear equality constraint targets (no relaxation)*

- Real `eta`
  *constant used in etaSequence updates*

- Real `alphaEta`
  *power for etaSequence updates when updating penalty*

- Real `betaEta`
  *power for etaSequence updates when updating multipliers*

- Real `etaSequence`
  *Lagrangian updates (refer to Conn, Gould, and Toint, section 14.4).*

### 8.144.1 Detailed Description

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

These minimizers use a `SurrogateModel` to perform optimization based either on local trust region methods or global updating methods.

### 8.144.2 Member Function Documentation

#### 8.144.2.1 `void run ()` [inline, protected, virtual]

and may contain pre/post steps in lieu of separate pre/post

_Actor_ supports a construct/initialize-run/run/finalize-run/destruct progression. This function is the virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from `Iterator`.

### 8.144.2.2 `void print_results (std::ostream & s)` [protected, virtual]

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

Reimplemented from `Iterator`.

### 8.144.2.3 `void update_lagrange_multipliers (const RealVector & fn_vals, const RealMatrix & fn_grads)` [protected]

initialize and update Lagrange multipliers for basic Lagrangian

For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through solution of a nonnegative linear least squares problem.

### 8.144.2.4 `void update_augmented_lagrange_multipliers (const RealVector & fn_vals)` [protected]

initialize and update the Lagrange multipliers for augmented Lagrangian

For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through solution of a nonnegative linear least squares problem.

### 8.144.2.5 `bool update_filter (const RealVector & fn_vals)` [protected]

update a filter from a set of function values

Update the sbFilter with fn_vals if new iterate is non-dominated.

### 8.144.2.6 `Real lagrangian_merit (const RealVector & fn_vals, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts)` [protected]

compute a Lagrangian function from a set of function values

The Lagrangian function computation sums the objective function and the Lagrange multiplier terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with \( g \leq 0 \) and \( h = 0 \). The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

### 8.144.2.7 `Real augmented_lagrangian_merit (const RealVector & fn_vals, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts)` [protected]

compute an augmented Lagrangian function from a set of function values

The Rockafellar augmented Lagrangian function sums the objective function, Lagrange multiplier terms for inequality/equality constraints, and quadratic penalty terms for inequality/equality constraints. This implementation
follows the convention in Vanderplaats with $g \leq 0$ and $h \leq 0$. The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

### 8.144.2.8 Real penalty_merit (const RealVector & fn_vals) [protected]

compute a penalty function from a set of function values

The penalty function computation applies a quadratic penalty to any constraint violations and adds this to the objective function(s) $p = f + r_p \cdot cv$.

### 8.144.2.9 Real objective (const RealVector & fn_vals) [protected]

compute a composite objective value from one or more objective functions

The composite objective computation sums up the contributions from one of more objective functions using the multiobjective weights.

### 8.144.2.10 void objective_gradient (const RealVector & fn_vals, const RealMatrix & fn_grads, RealVector & obj_grad) [protected]

compute the gradient of the composite objective function

The composite objective gradient computation sums up the contributions from one of more objective function gradients using the multiobjective weights.

### 8.144.2.11 Real constraint_violation (const RealVector & fn_vals, const Real & constraint_tol) [protected]

compute the constraint violation from a set of function values

Compute the quadratic constraint violation defined as $cv = g^+^T g^+ + h^+^T h^+$. This implementation supports equality constraints and 2-sided inequalities. The constraint_tol allows for a small constraint infeasibility (used for penalty methods, but not Lagrangian methods).

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

- SurrBasedMinimizer.H
- SurrBasedMinimizer.C
8.145 SurrogateDataPoint Class Reference

for defining a "truth" data point.

Public Member Functions

- **SurrogateDataPoint ()**
  default constructor

- **SurrogateDataPoint (const RealVector &x, const Real &fn_val, const RealVector &fn_grad, const RealSymMatrix &fn_hess)**
  standard constructor

- **SurrogateDataPoint (const SurrogateDataPoint &sdp)**
  copy constructor

- **~SurrogateDataPoint ()**
  destructor

- **SurrogateDataPoint & operator= (const SurrogateDataPoint &sdp)**
  assignment operator

- **bool operator== (const SurrogateDataPoint &sdp) const**
  equality operator

- **const RealVector & continuous_variables () const**
  return continuousVars

- **const Real & response_function () const**
  return responseFn

- **const RealVector & response_gradient () const**
  return responseGrad

- **const RealSymMatrix & response_hessian () const**
  return responseHess

- **bool is_null () const**
  function to check sdpRep (does this handle contain a body)

Private Attributes

- **SurrogateDataPointRep * sdpRep**
  pointer to the body (handle-body idiom)
8.145.1 Detailed Description

for defining a "truth" data point.

A list of these data points is contained in each Approximation instance (Approximation::currentPoints) and provides the data to build the approximation. A handle-body idiom is used to avoid excessive data copying overhead.

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

- DakotaApproximation.H
8.146 SurrogateDataPointRep Class Reference

or body, may be shared by multiple SurrogateDataPoint handle instances.

Private Member Functions

- SurrogateDataPointRep (const RealVector &x, const Real &fn_val, const RealVector &fn_grad, const RealSymMatrix &fn_hess)  
  constructor

- ~SurrogateDataPointRep ()  
  destructor

Private Attributes

- RealVector continuousVars  
  continuous variables

- Real responseFn  
  truth response function value

- RealVector responseGrad  
  truth response function gradient

- RealSymMatrix responseHess  
  truth response function Hessian

- int referenceCount  
  number of handle objects sharing sdpRep

Friends

- class SurrogateDataPoint  
  the handle class can access attributes of the body class directly

8.146.1 Detailed Description

or body, may be shared by multiple SurrogateDataPoint handle instances.
The SurrogateDataPoint/SurrogateDataPointRep pairs utilize a handle-body idiom (Coplien, Advanced C++). 
The documentation for this class was generated from the following file:
• DakotaApproximation.H
8.147 SurrogateModel Class Reference

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).

Inheritance diagram for SurrogateModel::

```
+----------+          +----------+
| Model    |          | SurrogateModel |
| SurrogateModel
| DataFitSurrModel | HierarchSurrModel |
```

Protected Member Functions

- **SurrogateModel** (ProblemDescDB &problem_db)
  
  * constructor

- **SurrogateModel** (ParallelLibrary &parallel_lib, const std::pair< short, short > &view, const Sizet2DArray &vars_comps, const ActiveSet &set, const String &corr_type, short corr_order)
  
  * alternate constructor

- **~SurrogateModel** ()

  * destructor

- **Model & subordinate_model** ()

  * return truth_model()

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

  * agreement with truth_response

- null void **apply_correction** (Response &approx_response, const RealVector &c_vars, bool quiet_flag=false)

  * apply the correction computed in compute_correction() to approx_response

- null void **auto_correction** (bool correction_flag)

  * sets autoCorrection to on (true) or off (false)

- null bool **auto_correction** ()

  * returns autoCorrection setting

- null void **check_submodel_compatibility** (const Model &sub_model)

  * HierarchSurrModel::highFidelityModel

- null bool **force_rebuild** ()
forced based on changes in the inactive data

- void asv_mapping (const ShortArray &orig_asv, ShortArray &actual_asv, ShortArray &approx_asv, bool build_flag)
  distributes the incoming orig_asv among actual_asv and approx_asv

- void asv_mapping (const ShortArray &actual_asv, const ShortArray &approx_asv, ShortArray &combined_asv)
  reconstitutes a combined_asv from actual_asv and approx_asv

- void response_mapping (const Response &actual_response, const Response &approx_response, Response &combined_response)
  overlays actual_response and approx_response to update combined_response

**Protected Attributes**

- bool mixedResponseSet
  flag for mixed approximate/actual responses

- IntSet surrogateFnIndices
  subset that is approximated

- IntResponseMap surrResponseMap
  derived_synchronize_nowait() functions

- IntRealVectorMap rawCVarsMap
  not contain lower level variables sets from finite differencing.

- IntIntMap truthIdMap
  DataFitSurrModel/HierarchSurrModel ids.

- IntIntMap surrIdMap
  DataFitSurrModel/HierarchSurrModel ids.

- IntResponseMap cachedApproxRespMap
  portions were still pending.

- String correctionType
  approximation correction approach to be used: additive or multiplicative

- short correctionOrder
  approximation correction order to be used: 0, 1, or 2

- bool autoCorrection
  and HierarchSurrModel approximate response computations
- bool correctionComputed
  and is available for application

- size_t approxBuilds
  number of calls to build_approximation()

- bool surrogateBypass
  on the underlying truth model.

- RealVector referenceCLBnds
  approximation is built; used to detect when a rebuild is required.

- RealVector referenceCUBnds
  approximation is built; used to detect when a rebuild is required.

- IntVector referenceDILBnds
  approximation is built; used to detect when a rebuild is required.

- IntVector referenceDIUBnds
  approximation is built; used to detect when a rebuild is required.

- RealVector referenceDRLBnds
  approximation is built; used to detect when a rebuild is required.

- RealVector referenceDRUBnds
  approximation is built; used to detect when a rebuild is required.

- RealVector referenceICVars
  rebuild is required.

- IntVector referenceIDIVars
  a rebuild is required.

- RealVector referenceIDRVars
  a rebuild is required.

Private Member Functions

- void apply_additive_correction (RealVector &alpha_corrected_fns, RealMatrix &alpha_corrected_grads, RealSymMatrixArray &alpha_corrected_hessians, const RealVector &c-vars, const ActiveSet &set)
  internal convenience function for applying additive corrections
void apply_multiplicative_correction (RealVector &beta_corrected_fns, RealMatrix &beta_corrected_grads, RealSymMatrixArray &beta_corrected_hessians, const RealVector &c_vars, const ActiveSet &set)

  **internal convenience function for applying multiplicative corrections**

**Private Attributes**

- **bool badScalingFlag**
  corrections; triggers an automatic switch to additive corrections

- **bool combinedFlag**
  flag indicating the combination of additive/multiplicative corrections

- **bool computeAdditive**
  flag indicating the need for additive correction calculations

- **bool computeMultiplicative**
  flag indicating the need for multiplicative correction calculations

- **RealVector addCorrFns**
  high and low fidelity model values at x=x_center.

- **RealMatrix addCorrGrads**
  high/low function difference at x=x_center.

- **RealSymMatrixArray addCorrHessians**
  high/low function difference at x=x_center.

- **RealVector multCorrFns**
  high fidelity to low fidelity model values at x=x_center.

- **RealMatrix multCorrGrads**
  of the high/low function ratio at x=x_center.

- **RealSymMatrixArray multCorrHessians**
  of the high/low function ratio at x=x_center.

- **RealVector combineFactors**
  correction instead of a strictly local correction.

- **RealVector correctionCenterPt**
  (x - x_c) terms in 1st-/2nd-order corrections.

- **RealVector correctionPrevCenterPt**
  copy of correctionCenterPt from the previous correction cycle
8.147 SurrogateModel Class Reference

- RealVector `approxFnsCenter`
  unavailable when applying 1st-/2nd-order multiplicative corrections.

- RealVector `approxFnsPrevCenter`
  copy of `approxFnsCenter` from the previous correction cycle

- RealMatrix `approxGradsCenter`
  unavailable when applying 1st-/2nd-order multiplicative corrections.

- RealVector `truthFnsCenter`
  Truth function values at the current correction point.

- RealVector `truthFnsPrevCenter`
  copy of `truthFnsCenter` from the previous correction cycle

- Variables `subModelVars`
  among differing variable views in `force_rebuild()`

- Constraints `subModelCons`
  among differing variable views in `force_rebuild()`

8.147.1 Detailed Description

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).
The SurrogateModel class provides common functions to derived classes for computing and applying corrections
to approximations.

8.147.2 Member Function Documentation

8.147.2.1 void compute_correction (const Response & truth_response, const Response & approx_response, const RealVector & c_vars) [protected, virtual]

agreement with truth_response

Compute an additive or multiplicative correction that corrects the approx_response to have 0th-order consistency
(matches values), 1st-order consistency (matches values and gradients), or 2nd-order consistency (matches values,
gradients, and Hessians) with the truth_response at a single point (e.g., the center of a trust region). The 0th-order,
1st-order, and 2nd-order corrections use scalar values, linear scaling functions, and quadratic scaling functions,
respectively, for each response function.

Reimplemented from Model.
8.147.2.2 bool force_rebuild () [protected, virtual]

forced based on changes in the inactive data

This function forces a rebuild of the approximation according to the sub-model variables view, the approximation type, and whether the active approximation bounds or inactive variable values have changed since the last approximation build.

Reimplemented from Model.

8.147.3 Member Data Documentation

8.147.3.1 bool autoCorrection [protected]

and HierarchSurrModel approximate response computations

SurrBasedOptStrategy must toggle this value since compute_correction() no longer automatically backs out an old correction.

8.147.3.2 size_t approxBuilds [protected]

number of calls to build_approximation()

used as a flag to automatically build the approximation if one of the derived compute_response functions is called prior to build_approximation().

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

- SurrogateModel.H
- SurrogateModel.C
8.148 SysCallAnalysisCode Class Reference

simulations using system calls.

Inheritance diagram for SysCallAnalysisCode::

```
AnalysisCode

SysCallAnalysisCode
```

Public Member Functions

- **SysCallAnalysisCode** (const ProblemDescDB &problem_db)
  
  *constructor*

- **~SysCallAnalysisCode** ()
  
  *destructor*

- **void spawn_evaluation** (const bool block_flag)
  
  *spawn a complete function evaluation*

- **void spawn_input_filter** (const bool block_flag)
  
  *spawn the input filter portion of a function evaluation*

- **void spawn_analysis** (const int &analysis_id, const bool block_flag)
  
  *spawn a single analysis as part of a function evaluation*

- **void spawn_output_filter** (const bool block_flag)
  
  *spawn the output filter portion of a function evaluation*

8.148.1 Detailed Description

simulations using system calls.

SysCallAnalysisCode creates separate simulation processes using the C system() command. It utilizes Command-Shell to manage shell syntax and asynchronous invocations.

8.148.2 Member Function Documentation

8.148.2.1 **void spawn_evaluation** (const bool block_flag)

spawn a complete function evaluation
Put the \texttt{SysCallAnalysisCode} to the shell. This function is used when all portions of the function evaluation (i.e., all analysis drivers) are executed on the local processor.

\textbf{8.148.2.2 \texttt{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.

\textbf{8.148.2.3 \texttt{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. 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.

\textbf{8.148.2.4 \texttt{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:

- \texttt{SysCallAnalysisCode.H}
- \texttt{SysCallAnalysisCode.C}
8.149 SysCallApplicInterface Class Reference

using system calls.

Inheritance diagram for SysCallApplicInterface:

```
   Interface
      |            |
      | ApplicationInterface
      |                   |
      | SysCallApplicInterface
```

Public Member Functions

- **SysCallApplicInterface** (const ProblemDescDB &problem_db)
  - constructor

- **∼SysCallApplicInterface** ()
  - destructor

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

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

- void derived_synch (PRPQueue &prp_queue)
- void derived_synch_nowait (PRPQueue &prp_queue)
- int derived_synchronous_local_analysis (const int &analysis_id)
- const StringArray &analysis_drivers () const
  - retrieve the analysis drivers specification for application interfaces

- const AnalysisCode * analysis_code () const
  - return AnalysisCode::fileNameMap when defined for derived Interface class

Private Member Functions

- void spawn_application (const bool block_flag)
  - and output filter. Called from derived_map() & derived_map_asynch().

- void derived_synch_kernel (PRPQueue &prp_queue)
  - derived_synch_nowait()
• bool system_call_file_test (const String &root_file)
  *the necessary results file(s)*

**Private Attributes**

• SysCallAnalysisCode sysCallSimulator
  *to a CommandShell in various combinations*

• IntSet sysCallSet
  *system call evaluations*

• IntShortMap failCountMap
  *map linking function evaluation id's to number of response read failures*

### 8.149.1 Detailed Description

using system calls.

SysCallApplicInterface uses a SysCallAnalysisCode object for performing simulation invocations.

### 8.149.2 Member Function Documentation

**8.149.2.1 void derived_synch (PRPQueue & prp_queue) [inline, virtual]**

Check for completion of active asynch jobs (tracked with sysCallSet). Wait for at least one completion and complete all jobs that have returned. This satisfies a "fairness" principle, in the sense that a completed job will _always_ be processed (whereas accepting only a single completion could always accept the same completion - the case of very inexpensive fn. evls. - and starve some servers).

Reimplemented from ApplicationInterface.

**8.149.2.2 void derived_synch_nowait (PRPQueue & prp_queue) [inline, virtual]**

Check for completion of active asynch jobs (tracked with sysCallSet). Make one pass through sysCallSet & complete all jobs that have returned.

Reimplemented from ApplicationInterface.

**8.149.2.3 int derived_synchronous_local_analysis (const int & analysis_id) [inline, virtual]**

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().

Reimplemented from ApplicationInterface.

The documentation for this class was generated from the following files:
- SysCallApplicInterface.H
- SysCallApplicInterface.C
8.150 TANA3Approximation Class Reference

approximation (a multipoint approximation).

Inheritance diagram for TANA3Approximation::

```
Approximation

+-----------------------------
|                            |
+-----------------------------

TANA3Approximation
```

Public Member Functions

- **TANA3Approximation ()**
  
  *default constructor*

- **TANA3Approximation (const ProblemDescDB &problem_db, const size_t &num_vars)**
  
  *standard constructor*

- **~TANA3Approximation ()**
  
  *destructor*

Protected Member Functions

- **int min_coefficients () const**
  
  *build the derived class approximation type in numVars dimensions*

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

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

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

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

- **void clear_current ()**
Private Member Functions

- void find_scaled_coefficients ()
  compute TANA coefficients based on scaled inputs

- void offset (const RealVector &x, RealVector &s)
  based on minX, apply offset scaling to x to define s

Private Attributes

- RealVector pExp
  vector of exponent values

- RealVector minX
  vector of minimum parameter values used in scaling

- RealVector scX1
  vector of scaled x1 values

- RealVector scX2
  vector of scaled x2 values

- Real H
  the scalar Hessian value in the TANA-3 approximation

8.150.1 Detailed Description

approximation (a multipoint approximation).

The TANA3Approximation class provides a multipoint approximation based on matching value and gradient data from two points (typically the current and previous iterates) in parameter space. It forms an exponential approximation in terms of intervening variables.

8.150.2 Member Function Documentation

8.150.2.1 void clear_current () [inline, protected, virtual]

Redefine default implementation to support history mechanism.
Reimplemented from Approximation.

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

- TANA3Approximation.H
- TANA3Approximation.C
8.151 TaylorApproximation Class Reference

series (a local approximation).

Inheritance diagram for TaylorApproximation::

```
Approximation
   |
   v
TaylorApproximation
```

Public Member Functions

- **TaylorApproximation ()**
  *default constructor*

- **TaylorApproximation (ProblemDescDB &problem_db, const size_t &num_vars)**
  *standard constructor*

- **~TaylorApproximation ()**
  *destructor*

Protected Member Functions

- **int min_coefficients () const**
  *build the derived class approximation type in numVars dimensions*

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

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

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

- **const RealSymMatrix & get_hessian (const RealVector &x)**
  *retrieve the approximate function Hessian for a given parameter vector*
8.151 TaylorApproximation Class Reference

8.151.1 Detailed Description

series (a local approximation).

The TaylorApproximation class provides a local approximation based on data from a single point in parameter space. It uses a first- or second-order Taylor series expansion: 

\[ f(x) = f(x_c) + \text{grad}(x_c)'(x - x_c) + (x - x_c)'\text{Hess}(x_c)(x - x_c)/2. \]

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

- TaylorApproximation.H
- TaylorApproximation.C
8.152 TrackerHTTP Class Reference

curl library

Public Member Functions

- TrackerHTTP ()
  default constructor is allowed, but doesn’t track methods used

- TrackerHTTP (ProblemDescDB &problem_db)
  standard constructor with ProblemDescDB

- ~TrackerHTTP ()
  destructor to free handles

- void post_start ()
  post the start of an analysis and archive start time

- void post_finish (unsigned runtime=0)
  post the completion of an analysis including elapsed time

Private Member Functions

- void initialize ()
  shared initialization functions across constructors

- void url_add_field (std::stringstream &url, const char *keyword, const std::string &value, bool delimit=true)
  set delimit = false to omit the &

- void build_default_data (std::stringstream &url, time_t &rawtime, const String &mode)
  construct URL with shared information for start/finish

- void send_data_using_get (std::string urltopost)
  whole url including location&fields

- void send_data_using_post (std::string datatopost)
  separate location and query; datatopost="name=daniel&project=curl"

- void populate_method_list (ProblemDescDB &problem_db)
  extract list of methods from problem database

- std::string get_uid ()
  get the real user ID
- std::string `get_username()`
  
  *get the username as reported by the environment*

- std::string `get_hostname()`
  
  *get the system hostname*

- std::string `get_os()`
  
  *get the operating system*

- std::string `get_datetime(time_t rawtime)`
  
  *get the date and time as a string YYYYMMDDHHMMSS*

### Private Attributes

- CURL * `curlPtr`
  
  *pointer to the curl handler instance*

- FILE * `devNull`
  
  *pointer to /dev/null*

- std::string `trackerLocation`
  
  *base URL for the tracker*

- std::string `proxyLocation`
  
  *(unlike default CURL behavior)*

- long `timeoutSeconds`
  
  *seconds until the request will timeout (may have issues with signals)*

- std::stringstream `methodList`
  
  *list of active methods*

- std::string `dakotaVersion`
  
  *DAKOTA version.*

- time_t `startTime`
  
  *cached starting time in raw seconds*

- short `outputLevel`
  
  *verbosity control*
8.152.1 Detailed Description

curl library

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

- TrackerHTTP.H
- TrackerHTTP.C
8.153 Variables Class Reference

Base class for the variables class hierarchy.

Inheritance diagram for Variables::

```
Variables
   ↓
MergedVariables
   ↓
MixedVariables
```

Public Member Functions

- **Variables ()**
  
  default constructor

- **Variables (const ProblemDescDB &problem_db)**
  
  standard constructor

- **Variables (const std::pair< short, short > &view, const Sizet2DArray &vars_comps, bool minimal_data=true)**
  
  alternate constructor for instantiations on the fly

- **Variables (const Variables &vars)**
  
  copy constructor

- **virtual ~Variables ()**
  
  destructor

- **Variables operator= (const Variables &vars)**
  
  assignment operator

- **virtual const UIntArray & merged_discrete_ids () const**
  
  returns the list of discrete variables merged into a continuous array

- **virtual void read (std::istream &s)**
  
  read a variables object from an std::istream

- **virtual void write (std::ostream &s) const**
  
  write a variables object to an std::ostream

- **virtual void write_aprepro (std::ostream &s) const**
  
  write a variables object to an std::ostream in aprepro format

- **virtual void read_annotated (std::istream &s)**
virtual void write_annotated (std::ostream &s) const
write a variables object in annotated format to an std::ostream

virtual void read_tabular (std::istream &s)
read a variables object in tabular format from an istream

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

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

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

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

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

size_t tv () const
total number of vars

size_t cv () const
number of active continuous vars

size_t cv_start () const
start index of active continuous vars

size_t div () const
number of active discrete int vars

size_t div_start () const
start index of active discrete int vars

size_t drv () const
number of active discrete real vars

size_t drv_start () const
start index of active discrete real vars

size_t icv () const
number of inactive continuous vars
• `size_t icv_start()` const
  
  *start index of inactive continuous vars*

• `size_t idiv()` const
  
  *number of inactive discrete int vars*

• `size_t idiv_start()` const
  
  *start index of inactive discrete int vars*

• `size_t idrv()` const
  
  *number of inactive discrete real vars*

• `size_t idrv_start()` const
  
  *start index of inactive discrete real vars*

• `size_t acv()` const
  
  *total number of continuous vars*

• `size_t adiv()` const
  
  *total number of discrete integer vars*

• `size_t adrv()` const
  
  *total number of discrete real vars*

• `const Real & continuous_variable(const size_t &i)` const
  
  *return an active continuous variable*

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

• `void continuous_variable(const Real &c_var, const size_t &i)`
  
  *set an active continuous variable*

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

• `const int & discrete_int_variable(const size_t &i)` const
  
  *return an active discrete integer variable*

• `const IntVector & discrete_int_variables()` const
  
  *return the active discrete integer variables*

• `void discrete_int_variable(const int &di_var, const size_t &i)`
  
  *set an active discrete integer variable*
• **void discrete_int_variables** (const IntVector &di_vars)
  *set the active discrete integer variables*

• **const Real & discrete_real_variable** (const size_t &i) const
  *return an active discrete real variable*

• **const RealVector & discrete_real_variables** () const
  *return the active discrete real variables*

• **void discrete_real_variable** (const Real &dr_var, const size_t &i)
  *set an active discrete real variable*

• **void discrete_real_variables** (const RealVector &dr_vars)
  *set the active discrete real variables*

• **StringMultiArrayConstView continuous_variable_labels** () const
  *return the active continuous variable labels*

• **void continuous_variable_labels** (StringMultiArrayConstView cv_labels)
  *set the active continuous variable labels*

• **void continuous_variable_label** (const String &cv_label, const size_t &i)
  *set an active continuous variable label*

• **StringMultiArrayConstView discrete_int_variable_labels** () const
  *return the active discrete integer variable labels*

• **void discrete_int_variable_labels** (StringMultiArrayConstView div_labels)
  *set the active discrete integer variable labels*

• **void discrete_int_variable_label** (const String &div_label, const size_t &i)
  *set an active discrete integer variable label*

• **StringMultiArrayConstView discrete_real_variable_labels** () const
  *return the active discrete real variable labels*

• **void discrete_real_variable_labels** (StringMultiArrayConstView drv_labels)
  *set the active discrete real variable labels*

• **void discrete_real_variable_label** (const String &drv_label, const size_t &i)
  *set an active discrete real variable label*

• **StringMultiArrayConstView continuous_variable_types** () const
  *return the active continuous variable types*

• **StringMultiArrayConstView discrete_int_variable_types** () const
return the active discrete integer variable types

- **StringMultiArrayConstView** `discrete_real_variable_types()` const
  return the active discrete real variable types

- **UIntMultiArrayConstView** `continuous_variable_ids()` const
  return the active continuous variable position identifiers

- **const RealVector &** `inactive_continuous_variables()` const
  return the inactive continuous variables

- **void** `inactive_continuous_variables(const RealVector &ic_vars)`
  set the inactive continuous variables

- **const IntVector &** `inactive_discrete_int_variables()` const
  return the inactive discrete variables

- **void** `inactive_discrete_int_variables(const IntVector &idi_vars)`
  set the inactive discrete variables

- **const RealVector &** `inactive_discrete_real_variables()` const
  return the inactive discrete variables

- **void** `inactive_discrete_real_variables(const RealVector &idr_vars)`
  set the inactive discrete variables

- **StringMultiArrayConstView** `inactive_continuous_variable_labels()` const
  return the inactive continuous variable labels

- **void** `inactive_continuous_variable_labels(StringMultiArrayConstView ic_vars)`
  set the inactive continuous variable labels

- **StringMultiArrayConstView** `inactive_discrete_int_variable_labels()` const
  return the inactive discrete variable labels

- **void** `inactive_discrete_int_variable_labels(StringMultiArrayConstView idi_vars)`
  set the inactive discrete variable labels

- **StringMultiArrayConstView** `inactive_discrete_real_variable_labels()` const
  return the inactive discrete variable labels

- **void** `inactive_discrete_real_variable_labels(StringMultiArrayConstView idr_vars)`
  set the inactive discrete variable labels

- **UIntMultiArrayConstView** `inactive_continuous_variable_ids()` const
  return the inactive continuous variable position identifiers
• `const RealVector & all_continuous_variables () const`  
  returns a single array with all continuous variables

• `void all_continuous_variables (const RealVector &ac_vars)`  
  sets all continuous variables using a single array

• `void all_continuous_variable (const Real &ac_var, const size_t &i)`  
  set a variable within the all continuous array

• `const IntVector & all_discrete_int_variables () const`  
  returns a single array with all discrete variables

• `void all_discrete_int_variables (const IntVector &adi_vars)`  
  sets all discrete variables using a single array

• `void all_discrete_int_variable (const int &adi_var, const size_t &i)`  
  set a variable within the all discrete array

• `const RealVector & all_discrete_real_variables () const`  
  returns a single array with all discrete variables

• `void all_discrete_real_variables (const RealVector &adr_vars)`  
  sets all discrete variables using a single array

• `void all_discrete_real_variable (const Real &adr_var, const size_t &i)`  
  set a variable within the all discrete array

• `StringMultiArrayConstView all_continuous_variable_labels () const`  
  returns a single array with all continuous variable labels

• `void all_continuous_variable_labels (StringMultiArrayConstView acv_labels)`  
  sets all continuous variable labels using a single array

• `void all_continuous_variable_label (const String &acv_label, const size_t &i)`  
  set a label within the all continuous label array

• `StringMultiArrayConstView all_discrete_int_variable_labels () const`  
  returns a single array with all discrete variable labels

• `void all_discrete_int_variable_labels (StringMultiArrayConstView adiv_labels)`  
  sets all discrete variable labels using a single array

• `void all_discrete_int_variable_label (const String &adiv_label, const size_t &i)`  
  set a label within the all discrete label array
- `StringMultiArrayConstView all_discrete_real_variable_labels () const`  
  returns a single array with all discrete variable labels

- `void all_discrete_real_variable_labels (StringMultiArrayConstView adrv_labels)`  
  sets all discrete variable labels using a single array

- `void all_discrete_real_variable_label (const String &adrv_label, const size_t &i)`  
  set a label within the all discrete label array

- `StringMultiArrayConstView all_continuous_variable_types () const`  
  return all continuous variable types

- `StringMultiArrayConstView all_discrete_int_variable_types () const`  
  return all discrete variable types

- `StringMultiArrayConstView all_discrete_real_variable_types () const`  
  return all discrete variable types

- `UIntMultiArrayConstView all_continuous_variable_ids () const`  
  return all continuous variable position identifiers

- `Variables copy () const`  
  for use when a deep copy is needed (the representation is _not_ shared)

- `void reshape (const Sizet2DArray &vars_comps)`  
  variablesComponents

- `const std::pair< short, short > & view () const`  
  returns variablesView

- `std::pair< short, short > get_view (const ProblemDescDB &problem_db) const`  
  defines variablesView from problem_db attributes

- `void inactive_view (short view2)`  
  sets the inactive view based on higher level (nested) context

- `const String & variables_id () const`  
  returns the variables identifier string

- `const Sizet2DArray & variables_components () const`  
  returns the number of variables for each of the constitutive components

- `bool is_null () const`  
  function to check variablesRep (does this envelope contain a letter)
Protected Member Functions

- **Variables** (BaseConstructor, const ProblemDescDB &problem_db, const std::pair< short, short > &view)
  
  `derived class constructors - Coplien, p. 159`  

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

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

- virtual void `build_active_views` ()
  
  `construct active views of all variables arrays`

- virtual void `build_inactive_views` ()
  
  `construct inactive views of all variables arrays`

- void `build_views` ()
  
  `construct active/inactive views of all variables arrays`

- void `initialize_all_continuous_var_types_ids` (bool relax)
  
  `allContinuousVarIds`

- void `initialize_all_discrete_int_var_types_ids` ()
  
  `Convenience function for initializing allDiscreteIntVarTypes.`

- void `initialize_all_discrete_real_var_types_ids` ()
  
  `Convenience function for initializing allDiscreteRealVarTypes.`

Protected Attributes

- std::pair< short, short > variablesView
  
  `view enumerations`

- Sizet2DArray variablesComponents
  
  `design/uncertain/state (first index) by sub-type (second index)`

- RealVector allContinuousVars
  
  `array combining all of the continuous variables (design, uncertain, state)`

- IntVector allDiscreteIntVars
  
  `array combining all of the discrete integer variables (design, state)`

- RealVector allDiscreteRealVars
  
  `array combining all of the discrete real variables (design, state)`
• StringMultiArray allContinuousLabels
  (design, uncertain, state)

• StringMultiArray allDiscreteIntLabels
  array combining all of the discrete integer variable labels (design, state)

• StringMultiArray allDiscreteRealLabels
  array combining all of the discrete real variable labels (design, state)

• StringMultiArray allContinuousVarTypes
  array of variable types for all of the continuous variables

• StringMultiArray allDiscreteIntVarTypes
  array of variable types for all of the discrete integer variables

• StringMultiArray allDiscreteRealVarTypes
  array of variable types for all of the discrete real variables

• UIntMultiArray allContinuousVarIds
  array of position identifiers for the all continuous variables array

• bool minimalData
  var types, and var ids, as these add too much overhead

• size_t cvStart
  start index of active continuous variables within allContinuousVars

• size_t divStart
  start index of active discrete integer variables within allDiscreteIntVars

• size_t drvStart
  start index of active discrete real variables within allDiscreteRealVars

• size_t icvStart
  start index of inactive continuous variables within allContinuousVars

• size_t idivStart
  start index of inactive discrete integer variables w/i allDiscreteIntVars

• size_t idrvStart
  start index of inactive discrete real variables within allDiscreteRealVars

• size_t numCV
  number of active continuous variables
• `size_t numDIV`
  number of active discrete integer variables

• `size_t numDRV`
  number of active discrete real variables

• `size_t numICV`
  number of inactive continuous variables

• `size_t numIDIV`
  number of inactive discrete integer variables

• `size_t numIDRV`
  number of inactive discrete real variables

• `RealVector continuousVars`
  the active continuous variables array view

• `IntVector discreteIntVars`
  the active discrete integer variables array view

• `RealVector discreteRealVars`
  the active discrete real variables array view

• `RealVector inactiveContinuousVars`
  the inactive continuous variables array view

• `IntVector inactiveDiscreteIntVars`
  the inactive discrete integer variables array view

• `RealVector inactiveDiscreteRealVars`
  the inactive discrete real variables array view

### Private Member Functions

• `Variables * get_variables (const ProblemDescDB &problem_db)`
  correct letter class

• `Variables * get_variables (const std::pair< short, short > &view) const`
  and by `copy()` to instantiate a new letter class

• `void check_view_compatibility ()`
  perform sanity checks on `view.first` and `view.second` after update
Private Attributes

- **String idVariables**
  
  variables identifier string from the input file

- **Variables * variablesRep**
  
  pointer to the letter (initialized only for the envelope)

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

Friends

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

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

- **std::size_t hash_value(const Variables &vars)**
  
  hash_value

- **bool binary_equal_to(const Variables &vars1, const Variables &vars2)**
  
  binary_equal_to (since 'operator==' is not suitable for boost/hash_set)

8.153.1 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.

8.153.2 Constructor & Destructor Documentation

8.153.2.1 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.
8.153.2.2  **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.

8.153.2.3  **Variables** (const std::pair< short, short > & view, const Sizet2DArray & vars_comps, bool minimal_data = true)

alternate constructor for instantiations on the fly

This is the alternate envelope constructor for instantiations on the fly. This constructor executes get_variables(view), which invokes the default derived/base constructors, followed by a reshape() based on vars_-comps.

8.153.2.4  **Variables** (const Variables & vars)

copy constructor

Copy constructor manages sharing of variablesRep and incrementing of referenceCount.

8.153.2.5  ~**Variables** () [virtual]

destructor

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

8.153.2.6  **Variables** (BaseConstructor, const ProblemDescDB & problem_db, const std::pair< short, short > & view) [protected]

derived class constructors - Coplien, p. 139)

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

8.153.3  **Member Function Documentation**

8.153.3.1  **Variables** operator= (const Variables & vars)

assignment operator

8.153.2 Variables copy () const

for use when a deep copy is needed (the representation is _not_ shared)
Deep copies are used for history mechanisms such as bestVariables and data_pairs since these must catalogue copies (and should not change as the representation within currentVariables changes).

8.153.3 void build_views () [inline, protected]

construct active/inactive views of all variables arrays
= EMPTY)
= EMPTY)

8.153.4 Variables * get_variables (const ProblemDescDB & problem_db) [private]

correct letter class
Initializes variablesRep to the appropriate derived type, as given by problem_db attributes. The standard derived class constructors are invoked.

8.153.5 Variables * get_variables (const std::pair< short, short > & view) const [private]

and by copy() to instantiate a new letter class
Initializes variablesRep to the appropriate derived type, as given by view. The default derived class constructors are invoked.

8.153.4 Member Data Documentation

8.153.4.1 UIntMultiArray allContinuousVarIds [protected]

array of position identifiers for the all continuous variables array
These identifiers define positions of the all continuous variables array within the total variable sequence.
The documentation for this class was generated from the following files:

- DakotaVariables.H
- DakotaVariables.C

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
Chapter 9

DAKOTA File Documentation

9.1 dll_api.C File Reference

This file contains a DakotaRunner class, which launches DAKOTA.

Namespaces

- namespace Dakota

Functions

- void signal_init ()
  initialize signal handlers

- void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, char *logname)
  create and configure a new DakotaRunner, adding it to list of instances

- int DAKOTA_DLL_FN dakota_readInput (int id, char *dakotaInput)
  command DakotaRunner instance id to read from file dakotaInput

- void DAKOTA_DLL_FN dakota_get_variable_info (int id, char ***pVarNames, int *pNumVarNames, char ***pRespNames, int *pNumRespNames)
  return the variable and response names

- int DAKOTA_DLL_FN dakota_start (int id)
  command DakotaRunner instance id to start (plugin interface and run strategy)

- void DAKOTA_DLL_FN dakota_destroy (int id)
  delete Dakota runner instance id and remove from active list

- void DAKOTA_DLL_FN dakota_stop (int *id)
command DakotaRunner instance id to stop execution

- const char *DAKOTA_DLL_FN dakota_getStatus(int id)
  return current results output as a string

- int get_mc_ptr_int()
  get the DAKOTA pointer to ModelCenter

- void set_mc_ptr_int(int ptr_int)
  set the DAKOTA pointer to ModelCenter

- int get_dc_ptr_int()
  get the DAKOTA pointer to ModelCenter current design point

- void set_dc_ptr_int(int ptr_int)
  set the DAKOTA pointer to ModelCenter current design point

Variables

- PRPCache data_pairs
  contains all parameter/response pairs.

- std::map<int, DakotaRunner*> runners
  map from DakotaRunner id to instance

9.1.1 Detailed Description

This file contains a DakotaRunner class, which launches DAKOTA.

9.1.2 Function Documentation

9.1.2.1 void DAKOTA_DLL_FN dakota_stop(int *id)

command DakotaRunner instance id to stop execution

TODO: trick application to quit through the syscall interface or throw exception.
9.2 dll_api.h File Reference

API for DLL interactions.

Functions

- void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, char *logname)
  create and configure a new DakotaRunner, adding it to list of instances

- int DAKOTA_DLL_FN dakota_readInput (int id, char *dakotaInput)
  command DakotaRunner instance id to read from file dakotaInput

- int DAKOTA_DLL_FN dakota_start (int id)
  command DakotaRunner instance id to start (plugin interface and run strategy)

- void DAKOTA_DLL_FN dakota_destroy (int id)
  delete Dakota runner instance id and remove from active list

- void DAKOTA_DLL_FN dakota_stop (int *id)
  command DakotaRunner instance id to stop execution

- const char * DAKOTA_DLL_FN dakota_getStatus (int id)
  return current results output as a string

- int DAKOTA_DLL_FN get_mc_ptr_int ()
  get the DAKOTA pointer to ModelCenter

- void DAKOTA_DLL_FN set_mc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter

- int DAKOTA_DLL_FN get_dc_ptr_int ()
  get the DAKOTA pointer to ModelCenter current design point

- void DAKOTA_DLL_FN set_dc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter current design point

- void DAKOTA_DLL_FN dakota_get_variable_info (int id, char ***pVarNames, int *pNumVarNames, char ***pRespNames, int *pNumRespNames)
  return the variable and response names

9.2.1 Detailed Description

API for DLL interactions.
9.2.2 Function Documentation

9.2.2.1 void DAKOTA_DLL_FN dakota_stop (int * id)

command DakotaRunner instance id to stop execution
TODO: trick application to quit through the syscall interface or throw exception.
Contains the implementation of the JEGAOptimizer class.

Namespaces

- namespace Dakota
- namespace JEGA::Logging
- namespace eddy::utilities

Classes

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

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

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

Functions

- template<typename T> string asstring (const T &val)
  
  Creates a string from the argument val using an ostringstream.

9.3.1 Detailed Description

Contains the implementation of the JEGAOptimizer class.
9.4 JEGAOptimizer.H File Reference

Contains the definition of the JEGAOptimizer class.

Namespaces

- namespace JEGA
- namespace JEGA::Utilities
- namespace JEGA::FrontEnd
- namespace JEGA::Algorithms
- namespace Dakota

Classes

- class JEGAOptimizer

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

9.4.1 Detailed Description

Contains the definition of the JEGAOptimizer class.
9.5 library_mode.C File Reference

File containing a mock simulator main for testing DAKOTA in library mode

Functions

- void `nidr_set_input_string` (const char *)
  
  *Set input to NIDR via string argument instead of input file.*

- void `run_dakota_parse` (const char *dakota_input_file)
  
  *mode 1: parsing an input file.*

- void `run_dakota_data` ()
  
  *mode 2: direct Data class instantiation.*

- void `run_dakota_mixed` (const char *dakota_input_file)
  
  *mode 3: mixed parsing and direct updating*

- void `model_interface_plugins` (Dakota::ProblemDescDB &problem_db)

- int `main` (int argc, char *argv[])
  
  *A mock simulator main for testing DAKOTA in library mode.*

- static void `my_callback_function` (void *ptr)

9.5.1 Detailed Description

File containing a mock simulator main for testing DAKOTA in library mode

9.5.2 Function Documentation

9.5.2.1 void `run_dakota_parse` (const char *dakota_input_file)

mode 1: parsing an input file.

This function parses from an input file to define the ProblemDescDB data.

9.5.2.2 void `run_dakota_data` ()

mode 2: direct Data class instantiation.

Rather than parsing from an input file, this function populates Data class objects directly using a minimal specification and relies on constructor defaults and post-processing in post_process() to fill in the rest.
9.5.2.3  void run_dakota_mixed (const char *dakota_input_file)

mode 3: mixed parsing and direct updating

This function showcases multiple features. For parsing, either an input file (dakota_input_file != NULL) or a
default input string (dakota_input_file == NULL) are shown. This parsed input is then mixed with input from three
sources: (1) input from a user-supplied callback function, (2) updates to the DB prior to Strategy instantiation, (3)
updates directly to Iterators/Models following Strategy instantiation.

9.5.2.4  void model_interface_plugins (Dakota::ProblemDescDB & problem_db)

Iterate over models and plugin appropriate interface: serial rosenbrock or parallel textbook.

9.5.2.5  int main (int argc, char * argv[ ])

A mock simulator main for testing DAKOTA in library mode.

Uses alternative instantiation syntax as described in the library mode documentation within the Developers Man-
ual. Tests several problem specification modes: (1) run_dakota_parse: reads all problem specification data from
an input file (2) run_dakota_data: creates all problem specification from direct Data instance instantiations. (3)
run_dakota_mixed: a mixture of input parsing (by file or default string) and direct data updates, where the data
updates occur: (a) via the DB prior to Strategy instantiation, and (b) via Iterators/Models following Strategy
instantiation. Usage: dakota_library_mode [-m] [dakota.in]

9.5.2.6  static void my_callback_function (void *ptr)  [static]

Example of user-provided callback function to override input specified and managed by NIDR, e.g., from an input
deck.
9.6 main.C File Reference

file containing the main program for DAKOTA

Functions

- void start_dakota_heartbeat (int)
- void fpinit_ASL ()
- int main (int argc, char *argv[])

The main DAKOTA program.

9.6.1 Detailed Description

file containing the main program for DAKOTA

9.6.2 Function Documentation

9.6.2.1 void start_dakota_heartbeat (int)

Heartbeat function provided by not_executable.c; pass output interval in seconds, or -1 to use $DAKOTA_-HEARTBEAT

9.6.2.2 void fpinit_ASL ()

Floating-point initialization from AMPL: switch to 53-bit rounding if appropriate, to eliminate some cross-platform differences.

9.6.2.3 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.
9.7  restart_util.C File Reference

file containing the DAKOTA restart utility main program

Namespaces

- namespace Dakota

Functions

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

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

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

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

- void concatenate_restart (int argc, char **argv)
  concatenate multiple restart files

- int main (int argc, char *argv[])
  The main program for the DAKOTA restart utility.

9.7.1 Detailed Description

file containing the DAKOTA restart utility main program

9.7.2 Function Documentation

9.7.2.1 int main (int argc, char * argv[])

The main program for the DAKOTA restart utility.

Parse command line inputs and invoke the appropriate utility function (print_restart(), print_restart_tabular(), read_neutral(), repair_restart(), or concatenate_restart()).
Chapter 10

Recommended Practices for DAKOTA Development

10.1 Introduction

Common code development practices can be extremely useful in multiple developer environments. Particular styles for code components lead to improved readability of the code and can provide important visual cues to other developers.

Much of this recommended practices document is borrowed from the CUBIT mesh generation project, which in turn borrows its recommended practices from other projects. As a result, C++ coding styles are fairly standard across a variety of Sandia software projects in the engineering and computational sciences.

10.2 Style Guidelines

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

10.2.1 Class and variable styles

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

```cpp
class ClassName;
```

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

```cpp
double classMemberVariable;
```

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

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

const double CONSTANT_VALUE;

### 10.2.2 Function styles

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

```cpp
int function_name();
```

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

```cpp
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.:

```cpp
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.:

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

### 10.2.3 Miscellaneous

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

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

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

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

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

```c
void abort_handler(int code)
{
    int initialized = 0;
    MPI_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.:

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

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

```c
cout << "Numerical gradients using " << finiteDiffStepSize*100. << ". " << finiteDiffType << " differences\n" << methodSource << " finite difference routine." << endl;
```

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

### 10.3 File Naming Conventions

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

File names for C++ classes should, in general, use the same name as the class defined by the file. Exceptions include:
• with the introduction of the Dakota namespace, base classes which previously utilized prepended Dakota identifiers can now safely omit the identifiers. However, since file names do not have namespace protection from name collisions, they retain the prepended Dakota identifier. For example, a class previously named DakotaModel which resided in DakotaModel.[CH], is now Dakota::Model (class Model in namespace Dakota) residing in the same filenames. The retention of the previous filenames reduces the possibility of multiple instances of a Model.H causing problems. Derived classes (e.g., NestedModel) do not require a prepended Dakota identifier for either the class or file names.

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

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

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

• .C A class implementation file ends in the suffix .C. An implementation file contains the definitions of the members of the class.

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

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

10.4 Class Documentation Conventions

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

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

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

These tools are no longer used, so remaining comment blocks of this type are informational only and will not appear in the documentation generated by doxygen.
Chapter 11

Instructions for Modifying DAKOTA’s Input Specification

11.1 Modify dakota.input.ns pec

The master input specification resides in dakota.input.nspec in Dakota/src. The master input specification uses the following syntactic elements:

- () for required group specifications
- [] for optional specifications
- | for alternatives
- {} for functions to process keywords

...to express logical relationships. These syntactic elements can be used to express various dependency relationships in the input specification. It is recommended that you review the existing specification and have an understanding of the constructs in use before attempting to add new ones.

Warning:

- Do not skip this step. Attempts to modify the NIDR_keywds.H file in Dakota/src without using the NIDR table generator are very error-prone. Moreover, the input specification provides a reference to the allowable inputs of a particular executable and should be kept in synch with the parser files; modifying the parser files independent of the input specification creates, at a minimum, undocumented features.
- All keywords in dakota.input.nspec are lower case by convention. All user inputs are converted to lower case by the parser prior to keyword match testing, resulting in case insensitive parsing.
- Since the NIDR parser allows abbreviation of keywords, you must avoid adding a keyword that could be misinterpreted as an abbreviation for a different keyword within the same top-level keyword, such as "strategy" and "method". For example, adding the keyword "expansion" within the method specification would be a mistake if the keyword "expansion_factor" already was being used in this specification.
• The NIDR input is somewhat order-dependent, allowing the same keyword to be reused multiple times in the specification. This often happens with aliases, such as lower_bounds, upper_bounds and initial_point. Ambiguities are resolved by attaching a keyword to the most recently seen context in which it could appear, if such exists, or to the first relevant context that subsequently comes along in the input file. With the earlier IDR parser, non-exclusive specifications (those not in mutually exclusive blocks) were required to be unique. That is why there are such aliases for initial_point as cdv_initial_point and ddv_initial_point: so older input files can be used with no or fewer changes.

11.2 Rebuild NIDR_keywds.H

cd Dakota/packages/nidr
make

These steps regenerate NIDR_keywds.H and dakota.input.summary in the Dakota/src directory. As described in more detail in the next section, you must manually update NIDRProblemDescDB.C in Dakota/src to accord with changes to dakota.input.nspec. If you commit changes to a source repository, be sure to commit the updated Dakota/src/NIDR_keywds.H, Dakota/src/dakota.input.nspec, Dakota/src/dakota.input.summary, and your manually updated Dakota/src/NIDRProblemDescDB.C.

11.3 Update NIDRProblemDescDB.C in Dakota/src

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

{ startfcn, startdata, stopfcn, stopdata }

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

Most of the things within braces in dakota.input.nspec are invocations of macros defined in NIDRProblemDescDB.C. The macros simplify writing dakota.input.nspec and make it more readable. Most macro invocations refer to little structures defined in NIDRProblemDescDB.C, usually with the help of other macros, some of which have different definitions in different parts of NIDRProblemDescDB.C. When adding a keyword to dakota.input.nspec, you may need to add a structure definition or even introduce a new data type. NIDRProblemDescDB.C has sections corresponding to each top-level keyword. The top-level keywords are in alphabetical order, and most entities in the section for a top-level keyword are also in alphabetical order. While not required, it is probably good practice to maintain this structure, as it makes things easier to find.
Any integer, real, or string data associated with a keyword are provided to the keyword’s startfcn, whose second argument is a pointer to a Values structure, defined in header file nidr.h.

**Example 1:** if you added the specification:

```latex
[method_setting REAL {method_setting_start, &method_setting_details} ]
```

you would provide a function

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

in NIDRProblemDescDB.C. In this example, argument &method_setting_details would be passed as v, val->n (the number of values) would be 1 and *val->r would be the REAL value given for the method_setting keyword. The method_setting_start function would suitably store this value with the help of method_setting_details.

For some top-level keywords, g (the third argument to the startfcn and stopfcn) provides access to a relevant context. For example, method_start (the startfcn for the top-level method keyword) executes

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

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

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

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

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

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

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

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

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

11.4.1 Augment/update get_<data_type>() functions

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

```c
const RealVector& get_rdv(const String& entry_name);
```

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

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

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

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

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

```c
{"method_name.method_setting", P methodSetting},
```
inserted at the beginning of the RVdmo array shown above (since the name in the existing first entry, i.e., "nested.primary_response_mapping", comes alphabetically after "method_name.method_setting").

11.5 Update Corresponding Data Classes

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

11.5.1 Update the Data class header file

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

11.5.2 Update the .C file

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

11.6 Use get_<data_type>() Functions

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

```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., dataMethodIter, dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) guaranteed to be set correctly. Outside of the constructors, the database list nodes will correspond to the last set operation, and may not return data from the desired list node.
11.7 Update the Documentation

Doxygen comments should be added to the Data class headers for the new attributes, and the reference manual sections describing the portions of dakota.input.nspec that have been modified should be updated.
Chapter 12

Interfacing with DAKOTA as a Library

12.1 Introduction

Some users may be interested in linking the DAKOTA toolkit into another application for use as an algorithm library. While this is not the primary usage model for DAKOTA, certain facilities are in place to allow this type of integration.

As part of the normal DAKOTA build process, where Dakota/configure –prefix=PREFIX has been run prior to make and make install, a libdakota.a is created and a copy of it is placed in PREFIX/lib (PREFIX defaults to /usr/local/Dakota). This library contains all source files from Dakota/src excepting the main.C, restart_util.C, and library_mode.C main programs. This library may be linked with another application through inclusion of -ldakota on the link line. Library and header paths may also be specified using the -L and -I compiler options (using PREFIX/lib and PREFIX/include, respectively). Depending on the configuration used when building this library, other libraries for the vendor optimizers and vendor packages will also be needed to resolve DAKOTA symbols for DOT, NPSOL, OPT++, SGOPT, LHS, Epetra, etc. Copies of these libraries are also placed in Dakota/lib. A sample XML specification of library names and paths is also available in Dakota/examples/linked_interfaces/linkage_spec.

Warning:

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

Attention:

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

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

12.3 Comparison to main.C

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

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

ParallelLibrary parallel_lib(argc, argv);

is replaced with

ParallelLibrary parallel_lib;

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

parallel_lib.specify_outputs_restart(cmd_line_handler);

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

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

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

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

12.4 Problem database population

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

12.4.1 Input file parsing

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

In this approach, the main.C call to

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

would be replaced with its overloaded form

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

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

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

12.4.2 Data node insertion

This approach is more involved than the previous approach, but it allows the application to publish all needed data to DAKOTA’s database directly, thereby eliminating the need for the parsing of a separate DAKOTA input file. In this case, ProblemDescDB::manage_inputs() is not called. Rather, DataStrategy, DataMethod, DataModel,
DataVariables, DataInterface, and DataResponses objects are instantiated and populated with the desired problem data. These objects are then published to the problem database using ProblemDescDB::insert_node(), e.g.:

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

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

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

The data objects are populated with their default values upon instantiation, so only the non-default values need to be specified. Refer to the DataStrategy, DataMethod, DataModel, DataVariables, DataInterface, and DataResponses class documentation and source code for lists of attributes and their defaults.

The default strategy is single_method, which runs a single iterator on a single model, and the default model is single, so it is not necessary to instantiate and publish a DataStrategy or DataModel object if advanced multi-component capabilities are not required. Rather, instantiation and insertion of a single DataMethod, DataVariables, DataInterface, and DataResponses object is sufficient for basic DAKOTA capabilities.

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

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

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

### 12.4.3 Mixed mode

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

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

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

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

to provide a finer grain of control. The passed input file `dakota_input_file` must contain all required inputs. Since vector data like variable values/bounds/tags, linear/nonlinear constraint coefficients/bounds, etc. are optional, these potentially large vector specifications can be omitted from the input file. Only the variable/response counts, e.g.
12.4 Problem database population

method
  linear_inequality_constraints = 500
variables
  continuous_design = 1000
responses
  num_objective_functions = 1
  num_nonlinear_inequality_constraints = 100000

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

Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem_db.set("variables.continuous_design.initial_point", drv);

where the string identifiers are the same identifiers used when pulling information from the database using one of the get_<datatype>() functions (refer to the source code of ProblemDescDB.C for a full list). However, the supported ProblemDescDB::set() options are a restricted subset of the database attributes, focused on vector inputs that can be large scale.

If performing these updates within the constructor of a DirectApplicInterface extension/derivation (see Defining the direct application interface), then this code is sufficient since the database is unlocked, the active list nodes of the ProblemDescDB have been set for you, and the correct strategy/method/model/variables/interface/responses specification instance will get updated. The difficulty in this case stems from the order of instantiation. Since the Variables and Response instances are constructed in the base Model class, prior to construction of Interface instances in derived Model classes, database information related to Variables and Response objects will have already been extracted by the time the Interface constructor is invoked and the database update will not propagate.

Therefore, it is preferred to perform these operations at a higher level (e.g., within your main program), prior to Strategy instantiation and execution, such that instantiation order is not an issue. However, in this case, it is necessary to explicitly manage the list nodes of the ProblemDescDB using a specification instance identifier that corresponds to an identifier from the input file, e.g.:

problem_db.set_db_variables_node("MY_VARIABLES_ID");
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem_db.set("variables.continuous_design.initial_point", drv);

Alternatively, rather than setting just a single data node, all data nodes may be set using a method specification identifier:

problem_db.set_db_list_nodes("MY_METHOD_ID");

since the method specification is responsible for identifying a model specification, which in turn identifies variables, interface, and responses specifications. If hardwiring specification identifiers is undesirable, then

problem_db.resolve_top_method();

can also be used to deduce the active method specification and set all list nodes based on it. This is most appropriate in the case where only single specifications exist for method/model/variables/interface/responses. In each of these cases, setting list nodes unlocks the corresponding portions of the database, allowing set/get operations.
Once all direct database updates have been performed in this manner, calls to `ProblemDescDB::broadcast()` and `ProblemDescDB::post_process()` should be used on all processors. The former will broadcast a packed MPI buffer with the aggregated set of specification data from rank 0 to other processors, and the latter will post-process specification data to fill in any vector defaults that have not yet been provided through either file parsing or direct updates (Note: scalar defaults are handled in the Data class constructors). Refer to `run_dakota_mixed()` in `library_mode.C` for a complete example listing.

### 12.5 Instantiating the strategy

With the `ProblemDescDB` object populated with problem data, we may now instantiate the strategy.

```c++
// instantiate the strategy
Strategy selected_strategy(problem_db);
```

Following strategy construction, all MPI communicator partitioning has been performed and the `ParallelLibrary` instance may be interrogated for parallel configuration data. For example, the lowest level communicators in DAKOTA’s multilevel parallel partitioning are the analysis communicators, which can be retrieved using:

```c++
// retrieve the set of analysis communicators for simulation initialization:
// one analysis comm per ParallelConfiguration (PC), one PC per Model.
Array<MPI_Comm> analysis_comms = parallel_lib.analysis_intra_communicators();
```

These communicators can then be used for initializing parallel simulation instances, where the number of MPI communicators in the array corresponds to one communicator per `ParallelConfiguration` instance.

### 12.6 Defining the direct application interface

When employing a library interface to DAKOTA, it is frequently desirable to also use a direct interface between DAKOTA and the simulation. There are two approaches to defining this direct interface.

#### 12.6.1 Extension

The first approach involves extending the existing `DirectApplicInterface` class to support additional direct simulation interfaces. In this case, a new simulation interface function can be added to Dakota/src/DirectApplic-Interface.[CH] for the simulation of interest. If the new function will not be a member function, then the following prototype should be used in order to pass the required data:

```c++
int sim(const Dakota::Variables& vars, const Dakota::ActiveSet& set, Dakota::Response& response);
```

If the new function will be a member function, then this can be simplified to

```c++
int sim();
```
since the data access can be performed through the DirectApplicInterface class attributes.

This simulation can then be added to the logic blocks in DirectApplicInterface::derived_map_ac(). In addition, DirectApplicInterface::derived_map_if() and DirectApplicInterface::derived_map_of() can be extended to perform pre- and post-processing tasks if desired, but this is not required.

While this approach is the simplest, it has the disadvantage that the DAKOTA library may need to be recompiled when the simulation or its direct interface is modified. If it is desirable to maintain the independence of the DAKOTA library from the host application, then the following derivation approach should be employed.

### 12.6.2 Derivation

The second approach is to derive a new interface from DirectApplicInterface in order to redefine several virtual functions. A typical derived class declaration might be

```cpp
namespace SIM {

class SerialDirectApplicInterface: public Dakota::DirectApplicInterface
{
public:

    // Constructor and destructor
    SerialDirectApplicInterface(const Dakota::ProblemDescDB& problem_db);
    ~SerialDirectApplicInterface();

protected:

    // Virtual function redefinitions
    int derived_map_if(const Dakota::String& if_name);
    int derived_map_ac(const Dakota::String& ac_name);
    int derived_map_of(const Dakota::String& of_name);

private:

    // Data
}
}
```

where the new derived class resides in the simulation's namespace. Similar to the case of Extension, the DirectApplicInterface::derived_map_ac() function is the required redefinition, and DirectApplicInterface::derived_map_if() and DirectApplicInterface::derived_map_of() are optional.

The new derived interface object (from namespace SIM) must now be plugged into the strategy. In the simplest case of a single model and interface, one could use

```cpp
// retrieve the interface of interest
ModelList& all_models = problem_db.model_list();
Model& first_model = *all_models.begin();
Interface& interface = first_model.interface();
// plug in the new direct interface instance (DB does not need to be set)
interface.assign_rep(new SIM::SerialDirectApplicInterface(problem_db), false);
```

from within the Dakota namespace. In a more advanced case of multiple models and multiple interface plug-ins, one might use
// retrieve the list of Models from the Strategy
Modellist& models = problem_db.model_list();
// iterate over the Model list
for (ModellIter ml_iter = models.begin(); ml_iter != models.end(); ml_iter++) {
  Interface& interface = ml_iter->interface();
  if (interface.interface_type() == "direct" &&
    interface.analysis_drivers().contains("SIM") ) {
    // set the correct list nodes within the DB prior to new instantiations
    problem_db.set_db_model_nodes(ml_iter->model_id());
    // plug in the new direct interface instance
    interface.assign_rep(new SIM::SerialDirectApplicInterface(problem_db), false);
  }
}

In the case where the simulation interface instance should manage parallel simulations within the context of an MPI communicator, one should pass in the relevant analysis communicator(s) to the derived constructor. For the latter case of looping over a set of models, the simplest approach of passing a single analysis communicator would use code similar to

const ParallelLevels ea_level = ml_iter->parallel_configuration_iterator()->ea_parallel_level();
const MPI_Comm& analysis_comm = ea_level.server_intra_communicator();
interface.assign_rep(new SIM::ParallelDirectApplicInterface(problem_db, analysis_comm), false);

Since Models may be used in multiple parallel contexts and may therefore have a set of parallel configurations, a more general approach would extract an array of analysis communicators to allow initialization for each of the parallel configurations.

New derived direct interface instances inherit various attributes of use in configuring the simulation. In particular, the ApplicationInterface::parallelLib reference provides access to MPI communicator data (e.g., the analysis communicators discussed in Instantiating the strategy), DirectApplicInterface::analysisDrivers provides the analysis driver names specified by the user in the input file, and DirectApplicInterface::analysisComponents provides additional analysis component identifiers (such as mesh file names) provided by the user which can be used to distinguish different instances of the same simulation interface. It is worth noting that inherited attributes that are set as part of the parallel configuration (instead of being extracted from the ProblemDescDB) will be set to their defaults following construction of the base class instance for the derived class plug-in. It is not until run-time (i.e., within derived_map_if/derived_map_ac/derived_map_of) that the parallel configuration settings are repropagated to the plug-in instance. This is the reason that the analysis communicator should be passed in to the constructor of a parallel plug-in, if the constructor will be responsible for parallel application initialization.

12.7 Additional updates

As part of strategy instantiation, all problem specification data is extracted from ProblemDescDB as various objects are constructed. Therefore, any updates that need to be performed following strategy instantiation must be performed through direct set operations on the constructed objects. In the previous section, the process for updating the Interface object used within a Model was shown. To update other data such as variable values/bounds/tags or response bounds/targets/tags, refer to the set functions documented in Iterator and Model. As an example, the following code updates the active continuous variable values, which will be employed as the initial guess for certain classes of Iterators:

Modellist& all_models = problem_db.model_list();
Model& first_model = *all_models.begin();
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
first_model.continuous_variables(drv);
12.8 Executing the strategy

Finally, with simulation configuration and plug-ins completed, we execute the strategy:

```cpp
// run the strategy
selected_strategy.run_strategy();
```

12.9 Retrieving data after a run

After executing the strategy, final results can be obtained through the use of `Strategy::variables_results()` and `Strategy::response_results()`, e.g.:

```cpp
// retrieve the final parameter values
const Variables& vars = selected_strategy.variables_results();

// retrieve the final response values
const Response& resp = selected_strategy.response_results();
```

In the case of optimization, the final design is returned, and in the case of uncertainty quantification, the final statistics are returned.

12.10 Linking against the DAKOTA library

This section presumes DAKOTA has been compiled and installed to `PREFIX` using ‘make install’. While the DAKOTA build system offers the most up-to-date guidance for what libraries are needed to link against a particular version of DAKOTA, a typical case is presented here. Note that depending on how you configured DAKOTA, some of the following libraries may not be available (for example NPSOL, DOT, NLPQL) – check which appear in `$PREFIX/lib`. Also as of DAKOTA 4.2, the link process is not as sensitive to order of these libraries, with the possible exception of liblhs.a.

As of DAKOTA 5.0, `-levidence` is no longer required and `-lgsl` is optional (discouraged due to GPL), depending on how DAKOTA was configured.

```
DAKOTA_LIBS = -L$(PREFIX)/lib -ldakota -lteuchos -lpecos -ldfftpack -llhs \ 
  -lsurfpack -lconmin -lddace -ldot -lfsudace \ 
  -ljega -lcport -lnlplq -lnpsol -lopt -ipsuade -lnewmat \ 
  -lconmic -lquadtrat -lcolnly -lcolin -lpebbl \ 
  -mlutilib -l3po -lnappspack -lappspack -lconveyor -lshared \ 
  -lcdd -lamlpsolver -llhs -llapack -llblas
```

You may also need `funcadd0.o`, `-lfl` and, if linking with system-provided GSL, `-lgslcblas`. The AMPL solver library may require `-ldl`. If configuring with graphics, you will need to add:

```
-lplplotcxxd -lplplotd -lgd -lpng -ljpeg -lz -lfreetype -lrt -lDGraphics
```

as well as any system X libraries (partial list here):

```
-lXpm -lXm -lXt -lXmu -lXp -lXext -lX11 -lSM -lICE
```

We have experienced problems with the creation of `libamplsolver.a` on some platforms. Please use the DAKOTA mailing lists for help with any problems.
12.11 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 is now in production use within several Sandia and external simulation codes/frameworks.
Chapter 13

Performing Function Evaluations

Performing function evaluations is one of the most critical functions of the DAKOTA software. It can also be one of the most complicated, as a variety of scheduling approaches and parallelism levels are supported. This complexity manifests itself in the code through a series of cascaded member functions, from the top level model evaluation functions, through various scheduling routines, to the low level details of performing a system call, fork, or direct function invocation. This section provides an overview of the primary classes and member functions involved.

13.1 Synchronous function evaluations

For a synchronous (i.e., blocking) mapping of parameters to responses, an iterator invokes Model::compute_response() to perform a function evaluation. This function is all that is seen from the iterator level, as underlying complexities are isolated. The binding of this top level function with lower level functions is as follows:

- Model::compute_response() utilizes Model::derived_compute_response() for portions of the response computation specific to derived model classes.
- Model::derived_compute_response() directly or indirectly invokes Interface::map().
- Interface::map() utilizes ApplicationInterface::derived_map() for portions of the mapping specific to derived application interface classes.

13.2 Asynchronous function evaluations

For an asynchronous (i.e., nonblocking) mapping of parameters to responses, an iterator invokes Model::asynch_compute_response() multiple times to queue asynchronous jobs and then invokes either Model::synchronize() or Model::synchronize_nowait() to schedule the queued jobs in blocking or nonblocking fashion. Again, these functions are all that is seen from the iterator level, as underlying complexities are isolated. The binding of these top level functions with lower level functions is as follows:

- Model::asynch_compute_response() utilizes Model::derived_asynch_compute_response() for portions of the response computation specific to derived model classes.
• This derived model class function directly or indirectly invokes Interface::map() in asynchronous mode, which adds the job to a scheduling queue.

• Model::synchronize() or Model::synchronize_nowait() utilize Model::derived_synchronize() or Model::derived_synchronize_nowait() for portions of the scheduling process specific to derived model classes.

• These derived model class functions directly or indirectly invoke Interface::synch() or Interface::synch_nowait().

• For application interfaces, these interface synchronization functions are responsible for performing evaluation scheduling in one of the following modes:
  – asynchronous local mode (using ApplicationInterface::asynchronous_local_evaluations() or ApplicationInterface::asynchronous_local_evaluations_nowait())
  – message passing mode (using ApplicationInterface::self_schedule_evaluations() or ApplicationInterface::static_schedule_evaluations() on the iterator master and ApplicationInterface::serve_evaluations_synch() or ApplicationInterface::serve_evaluations_peer() on the servers)
  – hybrid mode (using ApplicationInterface::self_schedule_evaluations() or ApplicationInterface::static_schedule_evaluations() on the iterator master and ApplicationInterface::serve_evaluations_asynch() on the servers)

• These scheduling functions utilize ApplicationInterface::derived_map() and ApplicationInterface::derived_map_asynch() for portions of asynchronous job launching specific to derived application interface classes, as well as ApplicationInterface::derived_synch() and ApplicationInterface::derived_synch_nowait() for portions of job capturing specific to derived application interface classes.

13.3 Analyses within each function evaluation

The discussion above covers the parallelism level of concurrent function evaluations serving an iterator. For the parallelism level of concurrent analyses serving a function evaluation, similar schedulers are involved (ForkApplicInterface::synchronous_local_analyses(), ForkApplicInterface::asynchronous_local_analyses(), ApplicationInterface::self_schedule_analyses(), ApplicationInterface::serve_analyses_synch(), ForkApplicInterface::serve_analyses_asynch()) to support synchronous local, asynchronous local, message passing, and hybrid modes. Not all of the schedulers are elevated to the ApplicationInterface level since the system call and direct function interfaces do not yet support nonblocking local analyses (and therefore support synchronous local and message passing modes, but not asynchronous local or hybrid modes). Fork interfaces, however, support all modes of analysis parallelism.
Chapter 14

Software Tools for DAKOTA Development

14.1 Introduction

DAKOTA development relies on Subversion for revision control and the GNU Autotools for configuration management. This section lists these tools, where to acquire recommended versions, and how to configure them.

14.2 Subversion for Version Control

The DAKOTA project uses Subversion (http://subversion.tigris.org/) for software version control. To check DAKOTA out of the Subversion revision control system on development.sandia.gov, it may be necessary to install or upgrade the Subversion client on your system. We are presently using version 1.3.2 available from http://subversion.tigris.org/downloads/subversion-1.3.2.tar.gz or newer.

To configure and build Subversion from source on your machine, the following settings should be used, since DAKOTA is hosted as a FSFS-type repository and depends on the external acro which is stored in a repository requiring SSL certificate handling:

```
tar xzf subversion-1.3.2.tar.gz
cd subversion-1.3.2
./configure --prefix=$HOME/local --with-ssl --without-berkeley-db CFLAGS=-O2
cd neon
./configure --prefix=$HOME/local --enable-shared --with-ssl --without-berkeley-db CFLAGS=-O2
cd ..
made && make check && make -k install
```

The make command as specified will ensure that Subversion is only installed if it passes all its self-tests, as well as making sure that the client install works correctly. Under some conditions, the Subversion build will attempt to write to /usr/lib, even when a --prefix option is passed to ./configure. This error may be disregarded when building the Subversion client, hence the -k option.

Once Subversion is working, DAKOTA (including externals) can be checked out with the single command

```
svn checkout svn+ssh://development.sandia.gov/usr/local/svn/Dakota/trunk Dakota
```
If you experience server timeouts when SVN attempts to fetch external packages through a proxy server, you might need to make a change to your $HOME/.subversion/servers file (generated for you the first time you run svn) by adding

```
[global]
http-proxy-exceptions = localhost, *.intranet.mydomain.com
http-proxy-host = wwwproxy.mydomain.com
```

to the bottom of the file. You should no longer get server timeouts when getting acro from software.sandia.gov. If you find that checking these three packages out from software is unacceptably slow, you may add your hostname to the end of the http-proxy-exceptions line. Finally, svn will prompt you as to whether you wish to accept the SSL certificate from software; type 'p' for permanent.

To set the default editor for Subversion commits, you may add the following to .cshrc:

```
setenv EDITOR "xemacs -g 81X50"
```

Version Control with Subversion (http://svnbook.red-bean.com) is a great resource on SVN.

### 14.3 GNU Autotools for Configuration Management

DAKOTA uses the GNU Autotools (http://www.gnu.org/software/autoconf/) for configuration management. Developers are currently using the following versions:

1. m4-1.4.3 (http://ftp.gnu.org/gnu/m4/m4-1.4.3.tar.gz)
2. libtool-1.5.24 (http://ftp.gnu.org/gnu/libtool/libtool-1.5.24.tar.gz)
3. automake-1.9.6 (http://ftp.gnu.org/gnu/automake/automake-1.9.6.tar.gz)
4. autoconf-2.60 (http://ftp.gnu.org/gnu/autoconf/autoconf-2.60.tar.gz)

Building the tools in the order listed above should satisfy dependencies. For each PACKAGE the following build process should suffice:

```
tar xzf $PACKAGE.tar.gz
cd $PACKAGE
./configure --prefix=$HOME/local
make
[make check]
make install
```

(Make check is useful for debugging builds of these packages, but optional and does take considerable time for some packages.)
Index

~Approximation
  Dakota::Approximation, 183
~BasisPolynomial
  Dakota::BasisPolynomial, 208
~BiStream
  Dakota::BiStream, 214
~Constraints
  Dakota::Constraints, 252
~EffGlobalMinimizer
  Dakota::EffGlobalMinimizer, 325
~Interface
  Dakota::Interface, 373
~Iterator
  Dakota::Iterator, 389
~Model
  Dakota::Model, 478
~ProblemDescDB
  Dakota::ProblemDescDB, 659
~Strategy
  Dakota::Strategy, 718
~Variables
  Dakota::Variables, 774
  _initPts
  Dakota::JEGAOptimizer, 403
  _model
  Dakota::JEGAOptimizer::Evaluator, 411

A
  Dakota::CONMINOptimizer, 242
abort_handler_t
  Dakota, 86
accepts_multiple_points
  Dakota::JEGAOptimizer, 402
actualModel
  Dakota::DataFitSurrModel, 266
add_datapoint
  Dakota::Graphics, 353
allContinuousVarIds
  Dakota::Variables, 775
alpha_polynomial
  Dakota::BasisPolynomial, 210
alpha_stat
  Dakota::BasisPolynomial, 210
append_approximation
  Dakota::ApproximationInterface, 187, 188
  Dakota::DataFitSurrModel, 264
approx_subprob_constraint_eval
  Dakota::SurrBasedLocalMinimizer, 734
approx_subprob_objective_eval
  Dakota::SurrBasedLocalMinimizer, 734
approxBuilds
  Dakota::SurrogateModel, 750
Approximation
  Dakota::Approximation, 182, 183
APPSEvalMgr
  Dakota::APPSOptimizer, 190
APPSOptimizer
  Dakota::APPSOptimizer, 193
Array
  Dakota::Array, 197
assign_rep
  Dakota::Interface, 373
  Dakota::Iterator, 392
  Dakota::Model, 481
asstring
  Dakota, 87
asynchronous_local_analyses
  Dakota::ForkApplicInterface, 332
asynchronous_local_evaluations
  Dakota::ApplicationInterface, 175
asynchronous_local_evaluations_nowait
  Dakota::ApplicationInterface, 176
asynchronous_local_evaluations_static
  Dakota::ApplicationInterface, 176
augmented_lagrangian_merit
  Dakota::SurrBasedMinimizer, 739
autoCorrection
  Dakota::SurrogateModel, 750

B
  Dakota::CONMINOptimizer, 241
BAD_OPCODE
CtelRegexp, 257
BAD_PARAM
CtelRegexp, 257
BasisPolynomial
Dakota::BasisPolynomial, 208
begins
Dakota::String, 722
beta_polynomial
Dakota::BasisPolynomial, 210
beta_stat
Dakota::BasisPolynomial, 210
Bgen
Dakota, 152
BiStream
Dakota::BiStream, 213, 214
BoStream
Dakota::BoStream, 216, 217
build_approximation
Dakota::ApproximationInterface, 188
Dakota::DataFitSurrModel, 263
build_global
Dakota::DataFitSurrModel, 265
build_local_multipoint
Dakota::DataFitSurrModel, 265
build_views
Dakota::Constraints, 253
Dakota::Variables, 775
C
Dakota::CONMINOptimizer, 241
CAUVLbl
Dakota, 149
CEUVLbl
Dakota, 150
check_integration
Dakota::NonDSparseGrid, 594
check_status
Dakota::ForkAnalysisCode, 330
check_variables
Dakota::NonDIntegration, 550
checkForEqualityConstraints
Dakota::SurfpackApproximation, 726
clear_all
Dakota::Approximation, 184
clear_current
Dakota::Approximation, 184
Dakota::TANA3Approximation, 757
Clone
Dakota::JEGAOptimizer::Evaluator, 411
close_streams
Dakota::ParallelLibrary, 638
ColinPoint, 225
compute_correction
Dakota::SurrogateModel, 749
compute_statistics
Dakota::NonDExpansion, 534
concatenate_restart
Dakota, 89
conminInfo
Dakota::CONMINOptimizer, 239
constraint0_evaluator
Dakota::SNLLOptimizer, 710
constraint1_evaluator
Dakota::SNLLOptimizer, 710
constraint1_evaluator_gn
Dakota::SNLLLeastSq, 703
constraint2_evaluator_gn
Dakota::SNLLLeastSq, 704
constraint_violation
Dakota::SurfBasedMinimizer, 740
constraintMappingIndices
Dakota::CONMINOptimizer, 239
Dakota::DOTOptimizer, 321
constraintMappingMultipliers
Dakota::CONMINOptimizer, 240
Dakota::DOTOptimizer, 322
constraintMappingOffsets
Dakota::CONMINOptimizer, 240
Dakota::DOTOptimizer, 322
Constraints
Dakota::Constraints, 252
constraintValues
Dakota::CONMINOptimizer, 239
Dakota::DOTOptimizer, 321
contains
Dakota::List, 426
Dakota::String, 722
copy
Dakota::Constraints, 253
Dakota::Variables, 774
copy_results
Dakota::ResponseRep, 683
copy_results_partial
Dakota::ResponseRep, 683
create_plots_2d
Dakota::Graphics, 353
create_tabular_datastream
Dakota::Graphics, 353
CreateEvaluator
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dakota::JEGAOptimizer::EvaluatorCreator</td>
<td>412</td>
</tr>
<tr>
<td>CT</td>
<td>kw_11, 92</td>
</tr>
<tr>
<td>Dakota::CONMINOptimizer</td>
<td>kw_110, 116</td>
</tr>
<tr>
<td>CtelRegexp</td>
<td>kw_111, 117</td>
</tr>
<tr>
<td>BAD_OPCODE</td>
<td>kw_12, 92</td>
</tr>
<tr>
<td>BAD_PARAM</td>
<td>kw_112, 117</td>
</tr>
<tr>
<td>EXP_TOO_BIG</td>
<td>kw_113, 117</td>
</tr>
<tr>
<td>GOOD</td>
<td>kw_114, 117, 144</td>
</tr>
<tr>
<td>INDEX_MATCH</td>
<td>kw_115, 117</td>
</tr>
<tr>
<td>INDEX_RANGE</td>
<td>kw_116, 118</td>
</tr>
<tr>
<td>INT_ERROR</td>
<td>kw_117, 118</td>
</tr>
<tr>
<td>OUT_OF_MEM</td>
<td>kw_118, 118</td>
</tr>
<tr>
<td>STARPLUS_EMPTY</td>
<td>kw_119, 118</td>
</tr>
<tr>
<td>STARPLUS_NESTED</td>
<td>kw_120, 119</td>
</tr>
<tr>
<td>STARPLUS_NOTHING</td>
<td>kw_121, 119</td>
</tr>
<tr>
<td>TOO_MANY_PAR</td>
<td>kw_122, 119</td>
</tr>
<tr>
<td>TRAILING</td>
<td>kw_123, 119</td>
</tr>
<tr>
<td>UNMATCH_PAR</td>
<td>kw_124, 120</td>
</tr>
<tr>
<td>CtelRegexp</td>
<td>kw_126, 120</td>
</tr>
<tr>
<td>RStatus</td>
<td>kw_127, 120</td>
</tr>
<tr>
<td>DakFuncs0</td>
<td>kw_128, 120, 144</td>
</tr>
<tr>
<td>Dakota</td>
<td>kw_129, 121</td>
</tr>
<tr>
<td>Dakota</td>
<td>kw_13, 93</td>
</tr>
<tr>
<td>abort_handler_t</td>
<td>kw_130, 121</td>
</tr>
<tr>
<td>asstring</td>
<td>kw_131, 121</td>
</tr>
<tr>
<td>Bgen</td>
<td>kw_132, 121</td>
</tr>
<tr>
<td>CAUVLbl</td>
<td>kw_133, 121</td>
</tr>
<tr>
<td>CEUVLbl</td>
<td>kw_134, 122</td>
</tr>
<tr>
<td>concatenate_restart</td>
<td>kw_135, 122</td>
</tr>
<tr>
<td>DakFuncs0</td>
<td>kw_136, 122</td>
</tr>
<tr>
<td>DAUVLbl</td>
<td>kw_137, 122</td>
</tr>
<tr>
<td>DAURVLbl</td>
<td>kw_138, 122</td>
</tr>
<tr>
<td>DiscSetLbl</td>
<td>kw_139, 123</td>
</tr>
<tr>
<td>FIELD_NAMES</td>
<td>kw_14, 93</td>
</tr>
<tr>
<td>flush</td>
<td>kw_140, 123</td>
</tr>
<tr>
<td>getdist</td>
<td>kw_141, 123</td>
</tr>
<tr>
<td>getRmax</td>
<td>kw_142, 123</td>
</tr>
<tr>
<td>id_vars_exact_compare</td>
<td>kw_143, 124</td>
</tr>
<tr>
<td>kw_1</td>
<td>kw_144, 124, 144</td>
</tr>
<tr>
<td>kw_10</td>
<td>kw_145, 124, 144</td>
</tr>
<tr>
<td>kw_100</td>
<td>kw_146, 124</td>
</tr>
<tr>
<td>kw_101</td>
<td>kw_147, 125</td>
</tr>
<tr>
<td>kw_102</td>
<td>kw_148, 125</td>
</tr>
<tr>
<td>kw_103</td>
<td>kw_149, 125</td>
</tr>
<tr>
<td>kw_104</td>
<td>kw_15, 93</td>
</tr>
<tr>
<td>kw_105</td>
<td>kw_150, 125</td>
</tr>
<tr>
<td>kw_106</td>
<td>kw_151, 126</td>
</tr>
<tr>
<td>kw_107</td>
<td>kw_152, 126</td>
</tr>
<tr>
<td>kw_108</td>
<td>kw_153, 126</td>
</tr>
</tbody>
</table>
kw_154, 126
kw_155, 126
kw_156, 127
kw_157, 127
kw_158, 127
kw_159, 127
kw_16, 94
kw_160, 128, 145
kw_161, 128
kw_162, 128
kw_163, 128
kw_164, 129
kw_165, 129
kw_166, 129, 145
kw_167, 129
kw_168, 130
kw_169, 130
kw_17, 94
kw_170, 130
kw_171, 130
kw_172, 130
kw_173, 131, 145
kw_174, 131
kw_175, 131
kw_176, 131
kw_177, 132, 145
kw_178, 132
kw_179, 132
kw_18, 94, 141
kw_180, 135
kw_181, 133
kw_182, 133
kw_183, 133
kw_184, 134
kw_185, 134
kw_186, 134
kw_187, 134
kw_188, 135
kw_189, 135
kw_19, 95
kw_190, 135
kw_191, 135
kw_192, 136
kw_193, 136, 145
kw_194, 136, 146
kw_195, 136
kw_196, 137, 146
kw_197, 137
kw_198, 137, 146
kw_199, 137
kw_2, 90, 139
kw_20, 95
kw_200, 138
kw_201, 138
kw_202, 138
kw_203, 138
kw_204, 139
kw_206, 139
kw_207, 146
kw_208, 147
kw_209, 147
kw_21, 95
kw_210, 147
kw_211, 147
kw_212, 148
kw_213, 148
kw_214, 148
kw_215, 148
kw_216, 149
kw_22, 95
kw_23, 95
kw_24, 96
kw_25, 96
kw_26, 96
kw_27, 97
kw_28, 97
kw_29, 98
kw_3, 90, 140
kw_30, 98
kw_31, 98
kw_32, 99
kw_33, 99, 142
kw_34, 99
kw_35, 99
kw_36, 100
kw_37, 100
kw_38, 100
kw_39, 100
kw_4, 90, 140
kw_40, 101
kw_41, 101, 142
kw_42, 101
kw_43, 101
kw_44, 101
kw_45, 102
kw_46, 102
kw_47, 102
kw_48, 102
kw_49, 102  kw_93, 112
kw_50, 103  kw_94, 112
kw_51, 103  kw_95, 112, 143
kw_52, 103  kw_96, 112
kw_53, 103  kw_97, 113
kw_54, 104  kw_98, 113
kw_55, 104  kw_99, 113
kw_56, 104  lookup_by_val, 87, 88
kw_57, 104  mindist, 86
kw_58, 104, 142  mindistindex, 86
kw_59, 105  NUMBER_OF_FIELDS, 89
kw_60, 105, 142  perform_analysis, 87
kw_61, 105  print_restart, 88
kw_62, 105  print_restart_tabular, 88
kw_63, 105  read_neutral, 88
kw_64, 106  repair_restart, 88
kw_65, 106  set_compare, 87
kw_66, 106  start_grid_computing, 87
kw_67, 106  stop_grid_computing, 87
kw_68, 106, 142  var_mp_bgen, 151
kw_69, 107  var_mp_bgen_audi, 151
kw_70, 107  var_mp_bgen_audr, 151
kw_71, 107  var_mp_bgen_dis, 152
kw_72, 107  var_mp_bgen_eu, 152
kw_73, 107  var_mp_bndchk, 152
kw_74, 108, 143  Vlch, 150
kw_75, 108  VLS, 150
kw_76, 108  Dakota::ActiveSet, 155
kw_77, 108  Dakota::ActiveSet
kw_78, 108, 143  derivVarsVector, 157
Dakota::AnalysisCode, 158
kw_79, 109  requestVector, 157
Dakota::Analyzer, 163
kw_8, 91, 141  evaluate_parameter_sets, 166
kw_80, 109  print_results, 166
kw_81, 109  print_vbd, 166
kw_82, 109  var_based_decomp, 166
kw_83, 110  Dakota::ApplicationInterface, 167
kw_84, 110  Dakota::ApplicationInterface
kw_85, 110  asynchronous_local_evaluations, 175
kw_86, 110  asynchronous_local_evaluations_nowait, 176
kw_87, 110  asynchronous_local_evaluations_static, 176
kw_88, 111  duplication_detect, 174
kw_89, 111, 143  init_serial, 173
kw_9, 92, 141  map, 173
kw_90, 111  self_schedule_analyses, 174
kw_91, 111  self_schedule_evaluations, 175
kw_92, 111  serve_analyses_synch, 174
serve_evaluations, 174
serve_evaluations_asynch, 177
serve_evaluations_peer, 177
serve_evaluations_sync, 176
static_schedule_evaluations, 175
stop_evaluation_servers, 174
synch, 173
synch_nowait, 173
synchronous_local_evaluations, 176
Dakota::Approximation, 178
~Approximation, 183
Approximation, 182, 183
clear_all, 184
clear_current, 184
get_approx, 184
operator=, 183
Dakota::ApproximationInterface, 185
Dakota::ApproximationInterface
append_approximation, 187, 188
build_approximation, 188
functionSurfaces, 188
update_approximation, 187
Dakota::APPSValMgr, 189
Dakota::APPSValMgr
APPSEvalMgr, 190
isWaiting, 191
recv, 191
spawn, 191
Dakota::APPSoptimizer, 192
APPSoptimizer, 193
find_optimum, 194
initialize_variables_and_constraints, 194
set_apps_parameters, 194
Dakota::Array, 195
Array, 197
data, 197
operator T *, 197
operator=, 197
operator[], 197
Dakota::BaseConstructor, 199
Dakota::BasisPolyApproximation, 200
Dakota::BasisPolyApproximation
expansionCoeffGrads, 204
tensor_product_terms, 204
total_order_terms, 204
Dakota::BasisPolynomial, 205
Dakota::BasisPolynomial
~BasisPolynomial, 208
alpha_polynomial, 210
Dakota::CollaborativeHybridStrategy
Dakota::CommandLineHandler
Dakota::CommandShell
flush, 231
Dakota::ConcurrentStrategy, 232
Dakota::ConcurrentStrategy
pack_parameters_buffer, 233
pack_results_buffer, 234
unpack_parameters_buffer, 233
unpack_results_buffer, 234
Dakota::CONMINOptimizer, 235
A, 242
INDEX

B, 241
C, 241
conminInfo, 239
constraintMappingIndices, 239
constraintMappingMultipliers, 240
constraintMappingOffsets, 240
constraintValues, 239
CT, 241
DF, 242
G1, 241
G2, 241
IC, 242
ISC, 242
MS1, 241
N1, 240
N2, 240
N3, 240
N4, 240
N5, 240
optimizationType, 239
printControl, 239
S, 241
SCAL, 241
Dakota::Constraints, 243
~Constraints, 252
build_views, 253
Constraints, 252
copy, 253
get_constraints, 254
manage_linear Constraints, 253
operator=, 253
reshape, 253
Dakota::DataFitSurrModel, 258
Dakota::DataFitSurrModel
actualModel, 266
append_approximation, 264
build_approximation, 263
build_global, 265
build_local_multipoint, 265
derived_asynch compute_response, 262
derived_compute_response, 262
derived_init_communicators, 264
derived_synchronize, 263
derived_synchronize nowait, 263
evaluation id, 265
update_actual_model, 265
update_approximation, 263, 264
update_from_actual_model, 265
Dakota::DataInterface, 267
Dakota::DataMethod, 268
Dakota::DataMethodRep, 269
Dakota::DataModel, 281
Dakota::DataModelRep, 282
Dakota::DataResponses, 286
Dakota::DataResponsesRep, 287
Dakota::DataStrategy, 291
Dakota::DataStrategyRep, 292
Dakota::DataVariables, 295
Dakota::DataVariablesRep, 297
Dakota::DDACEDesignCompExp, 307
Dakota::DDACEDesignCompExp
~DDACEDesignCompExp, 309
derived_post_run, 309
pre_run, 309
resolve_samples_symbols, 310
Dakota::DirectApplicInterface, 311
Dakota::DirectApplicInterface
derived_map_ac, 317
derived_synchronous_local_analysis, 317
Dakota::DOTOptimizer, 318
constraintMappingIndices, 321
constraintMappingMultipliers, 322
constraintMappingOffsets, 322
constraintValues, 321
dotFSInfo, 320
dotInfo, 320
dotMethod, 321
intCntParmArray, 321
optimizationType, 321
printControl, 321
realCntParmArray, 321
Dakota::EffGlobalMinimizer, 323
Dakota::EffGlobalMinimizer
~EffGlobalMinimizer, 325
Dakota::EmbeddedHybridStrategy, 326
Dakota::ForkAnalysisCode, 329
Dakota::ForkAnalysisCode
check_status, 330
Dakota::ForkApplicInterface, 331
Dakota::ForkApplicInterface
asynchronous local analyses, 332
derived_synchronous_local_analysis, 332
fork_application, 332
serve_analyses Async, 333
synchronous local analyses, 333
Dakota::FSUDesignCompExp, 334
Dakota::FSUDesignCompExp
derived_post_run, 336

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
enforce_input_rules, 337
FSUDesignCompExp, 336
pre_run, 336
Dakota::FunctionCompare, 338
Dakota::GaussProcApproximation, 339
Dakota::GaussProcApproximation
GPmodel_apply, 344
Dakota::GenLaguerreOrthogPolynomial, 345
Dakota::GenLaguerreOrthogPolynomial
Dakota::GetLongOpt, 347
MandatoryValue, 349
OptionalValue, 348
Valueless, 348
Dakota::GetLongOpt
enroll, 349
GetLongOpt, 349
OptType, 348
parse, 349
retrieve, 349
usage, 350
Dakota::Graphics, 351
add_datapoint, 353
create_plots_2d, 353
create_tabular_datastream, 353
new_dataset, 353
show_data_3d, 353
Dakota::GridApplicInterface, 355
Dakota::GridApplicInterface
derived_synchronous_local_analysis, 356
Dakota::HermiteOrthogPolynomial, 358
Dakota::HierarchSurrModel, 360
Dakota::HierarchSurrModel
derived_asynch_compute_response, 363
derived_compute_response, 362
derived_synchronize, 363
derived_synchronize_nowait, 363
evaluation_id, 363
Dakota::HybridStrategy, 364
Dakota::Interface, 366
~Interface, 373
assign_rep, 373
get_interface, 374
Interface, 373
operator=, 373
rawResponseMap, 374
Dakota::InterpPolyApproximation, 375
Dakota::InterpPolyApproximation
get_mean, 378
get_mean_gradient, 378, 379
get_subsets, 380
get_variance, 379
get_variance_gradient, 379
lower_sets, 380
partial_variance, 380
partial_variance_integral, 380
polynomialBasis, 380
smolyakMultiIndex, 380
Dakota::Iter, 382
~Iter, 389
assign_rep, 392
derived_final_initialize_run, 392
derived_initialize_run, 392
derived_post_run, 392
fdGradStepSize, 393
fdHessByFnStepSize, 393
fdHessByGradStepSize, 393
finalize_run, 391
get_iterator, 393
initialize_graphics, 390
initialize_run, 390
Iter, 388, 389
operator=, 389
post_run, 391
pre_run, 390
print_results, 390
run, 390
run_iterator, 391
Dakota::JacobiOrthogPolynomial, 394
Dakota::JEGAOptimizer, 396
_initPts, 403
accepts_multiple_points, 402
find_optimum, 401
GetBestMOSolution, 400
GetBestSolution, 400
GetBestSO_solution, 400
initial_points, 402
JEGAOptimizer, 398
LoadAlgorithmConfig, 399
LoadDakotaResponses, 399
LoadProblemConfig, 399
LoadTheConstraints, 400
LoadTheDesignVariables, 399
LoadTheObjectiveFunctions, 400
LoadTheParameterDatabase, 399
resize_response_results_array, 401
resize_variables_results_array, 401
returns_multiple_points, 402
ToDoubleMatrix, 401
Dakota::JEGAOptimizer::Driver, 404
update_from_subordinate_model, 480
update_quasi_hessians, 483
update_response, 483
Dakota::Model::FDhelp, 484
Dakota::MPIPackBuffer, 485
Dakota::MPIUnpackBuffer, 488
Dakota::NCSUOptimizer, 491
NCUSOptimizer, 493
Dakota::NestedModel, 494
Dakota::NestedModel
derived_asynch_compute_response, 498
derived_compute_response, 498
derived_init_communicators, 499
derived_master_overload, 499
evaluation_id, 499
response_mapping, 499
subModel, 500
Dakota::NIDRProblemDescDB, 501
Dakota::NIDRProblemDescDB
derived_parse_inputs, 504
Dakota::NL2Res, 506
Dakota::NL2SOLLeastSq, 507
Dakota::NLPSOLPNormizer, 510
Dakota::NLSSOLLeastSq, 515
Dakota::NLSSOLLeastSq
NLSSOLLeastSq, 516
Dakota::NoDBaseConstructor, 517
Dakota::NonD, 518
Dakota::NonD
derived_finalize_run, 524
derived_initialize_run, 524
initialize_final_statistics, 524
initialize_random_variable_parameters, 525
initialize_random_variable_types, 525
initialize_random_variables, 524
run, 524
set_u_to_x_mapping, 525
vars_u_to_x_mapping, 525
Dakota::NonDAdaptImpSampling, 526
Dakota::NonDAdaptImpSampling
initialize, 529
NonDAdaptImpSampling, 528
Dakota::NonDBayesCal, 530
Dakota::NonDBayesCal
NonBayesCal, 531
quantify_uncertainty, 531
Dakota::NonDExpansion, 532
Dakota::NonDExpansion
compute_statistics, 534
Dakota::NonDGlobalEvidence, 535
Dakota::NonDGlobalInterval, 537
Dakota::NonDGlobalReliability, 541
Dakota::NonDGlobalSingleInterval, 544
Dakota::NonDIncrementLHSSampling, 546
Dakota::NonDIncrementLHSSampling
NonDIncrementLHSSampling, 547
quantify_uncertainty, 547
Dakota::NonDIntegration, 548
Dakota::NonDIntegration
check_variables, 550
initialize, 550
NonDIntegration, 549
Dakota::NonDInterval, 551
Dakota::NonDLHSEvidence, 554
Dakota::NonDLHSInterval, 556
Dakota::NonDLHSSampling, 558
Dakota::NonDLHSSampling
NonDLHSSampling, 559
quantify_uncertainty, 560
Dakota::NonDLHSSingleInterval, 561
Dakota::NonDLLocalEvidence, 563
Dakota::NonDLLocalInterval, 565
Dakota::NonDLLocalReliability, 568
Dakota::NonDLLocalReliability
dg_ds_eval, 573
initial_taylor_series, 572
initialize_class_data, 572
initialize_level_data, 572
initialize_mpp_search_data, 572
probability, 573
reliability, 573
update_level_data, 572
update_mpp_search_data, 572
update_pma_reliability_level, 573
Dakota::NonDLLocalSingleInterval, 574
Dakota::NonDPolynomialChaos, 576
Dakota::NonDQuadrature, 578
Dakota::NonDQuadrature
initialize, 579
NonDQuadrature, 579
sampling_reset, 579
Dakota::NonDReliability, 581
Dakota::NonDReliability
PMA_constraint_eval, 584
PMA_objective_eval, 584
RIA_constraint_eval, 583
RIA_objective_eval, 583
Dakota::NonDSampling, 585

DAKOTA Version 5.0 Developers Manual generated on May 7, 2010
Dakota::NonDSampling

get_parameter_sets, 589
NonDSampling, 588
sampling_reset, 589

Dakota::NonDSparseGrid, 590

Dakota::NonDSparseGrid

check_integration, 594
initialize, 594
isotropicSSG, 595
level_to_order_closed_exponential, 594
level_to_order_open_exponential, 594
level_to_order_open_linear, 594
NonDSparseGrid, 594
sampling_reset, 594

Dakota::NonDStochCollocation, 596

Dakota::NPSOLOptimizer, 597
NPSOLOptimizer, 599

Dakota::NumericGenOrthogPolynomial, 601

Dakota::Optimizer, 606

derived_finalize_run, 609
derived_initialize_run, 608
derived_post_run, 609
multi_objective_retrieve, 609
Optimizer, 608
primary_resp_recast, 609
print_results, 609
run, 608
weighted_sum, 609

Dakota::OrthogonalPolynomial, 611

Dakota::OrthogPolyApproximation, 613

Dakota::OrthogPolyApproximation

expectation, 619
get_mean, 617
get_mean_gradient, 618
get_variance, 618
get_variance_gradient, 618, 619
gradient_check, 620
integration, 619
regression, 619
sparse_grid_terms, 617

Dakota::ParallelConfiguration, 621

Dakota::ParallelLevel, 624

Dakota::ParallelLibrary, 627

Dakota::ParallelLibrary

close_streams, 638
increment_parallel_configuration, 638
init_communicators, 638

manage_outputs_restart, 638
ParallelLibrary, 637
resolve_inputs, 638
specify_outputs_restart, 637

Dakota::ParamResponsePair, 640

Dakota::ParamResponsePair
evalInterfaceIds, 643
ParamResponsePair, 642
read, 642
write, 642

Dakota::ParamStudy, 644

Dakota::ParamStudy
derived_post_run, 647
pre_run, 647

Dakota::ProblemDescDB, 651

Dakota::ProblemDescDB

~ProblemDescDB, 659
get_db, 660
manage_inputs, 659, 660
operator=, 659
parse_inputs, 660
post_process, 660
ProblemDescDB, 659

Dakota::PStudyDACE, 661

Dakota::PStudyDACE

print_results, 662
run, 662
volumetric_quality, 662

Dakota::PSUADEDesignCompExp, 664

Dakota::PSUADEDesignCompExp
derived_post_run, 666
enforce_input_rules, 666
pre_run, 666
PSUADEDesignCompExp, 666

Dakota::RecastBaseConstructor, 667

Dakota::RecastModel, 668

Dakota::RecastModel

initialize, 673
RecastModel, 673
update_from_sub_model, 673

Dakota::Response, 674
Response, 678

Dakota::ResponseRep, 679

Dakota::ResponseRep
copy_results, 683
copy_results_partial, 683
functionGradients, 684
read, 682, 683
read_annotated, 682
read_tabular, 682
reset, 684
reset_inactive, 684
reshape, 684
ResponseRep, 681
write, 682, 683
write_annotated, 682
write_tabular, 682
Dakota::SensAnalysisGlobal, 685
Dakota::SequentialHybridStrategy, 687
Dakota::SequentialHybridStrategy
extract_parameter_sets, 690
extract_results_sets, 690
merge_results_sets, 690
pack_parameters_buffer, 689
pack_results_buffer, 689
run_sequential, 690
run_sequential_adaptive, 690
unpack_parameters_buffer, 689
unpack_results_buffer, 689
Dakota::SingleMethodStrategy, 693
Dakota::SingleModel, 695
Dakota::SNLLBase, 698
Dakota::SNLLLeastSq, 701
Dakota::SNLLLeastSq
constraint1_evaluator_gn, 703
constraint2_evaluator_gn, 704
nlf2_evaluator_gn, 703
Dakota::SNLLOptimizer, 705
constraint0_evaluator, 710
constraint1_evaluator, 710
nlf0_evaluator, 709
nlf1_evaluator, 710
nlf2_evaluator, 710
SNLLOptimizer, 709
Dakota::SOLBase, 711
Dakota::Strategy, 714
~Strategy, 718
free_iterator, 720
get_strategy, 720
init_iterator, 719
init_iterator_parallelism, 719
operator=, 718
pack_parameters_buffer, 718
pack_results_buffer, 719
run_iterator, 719
schedule_iterators, 720
self_schedule_iterators, 720
serve_iterators, 720
Strategy, 717, 718
unpack_parameters_buffer, 719
unpack_results_buffer, 719
Dakota::String, 721
begins, 722
contains, 722
data, 723
ends, 723
lower, 722
operator const char *, 722
upper, 722
Dakota::SurfpackApproximation, 724
Dakota::SurfpackApproximation
checkForEqualityConstraints, 726
find_coefficients, 725
get_hessian, 726
surrogates_to_surf_data, 726
Dakota::SurrBasedGlobalMinimizer, 727
Dakota::SurrBasedLocalMinimizer, 729
Dakota::SurrBasedLocalMinimizer
approx_subprob_constraint_eval, 734
approx_subprob_objective_eval, 734
hard_convergence_check, 733
hom_constraint_eval, 735
hom_objective_eval, 734
minimize_surrogates, 733
tr_ratio_check, 734
update_penalty, 734
Dakota::SurrBasedMinimizer, 736
Dakota::SurrBasedMinimizer
augmented_lagrangian_merit, 739
constraintViolation, 740
lagrangian_merit, 739
objective, 740
objective_gradient, 740
penalty_merit, 740
print_results, 739
run, 738
update_augmented_lagrange_multipliers, 739
update_filter, 739
update_lagrange_multipliers, 739
Dakota::SurrogateDataPoint, 741
Dakota::SurrogateDataPointRep, 743
Dakota::SurrogateModel, 745
Dakota::SurrogateModel
approxBuilds, 750
autoCorrection, 750
compute_correction, 749
force_rebuild, 749
Dakota::SysCallAnalysisCode, 751
Dakota::SysCallAnalysisCode
spawn_analysis, 752
spawn_evaluation, 751
spawn_input_filter, 752
spawn_output_filter, 752
Dakota::SysCallApplicInterface, 753
Dakota::SysCallApplicInterface
derived_sync, 754
derived_sync_nowait, 754
derived_synchronous_local_analysis, 754
Dakota::TANA3Approximation, 756
clear_current, 757
Dakota::TaylorApproximation, 758
Dakota::TrackerHTTP, 760
Dakota::Variables, 763
~Variables, 774
allContinuousVarIds, 775
build_views, 775
copy, 774
get_variables, 775
operator=, 774
Variables, 773, 774
dakota_stop
dll_api.C, 778
dll_api.h, 780
data
Dakota::Array, 197
Dakota::String, 723
DAUIVLbl
Dakota, 149
DAURVLbl
Dakota, 150
DDACEDesignCompExp
Dakota::DDACEDesignCompExp, 309
derivative_concurrency
Dakota::Model, 482
derived_asynch_compute_response
Dakota::DataFitSurrModel, 262
Dakota::HierarchSurrModel, 363
Dakota::NestedModel, 498
derived_compute_response
Dakota::DataFitSurrModel, 262
Dakota::HierarchSurrModel, 362
Dakota::NestedModel, 498
derived finalize_run
Dakota::Iterator, 392
Dakota::LeastSq, 420
Dakota::Minimizer, 438
Dakota::NonD, 524
Dakota::Optimizer, 609
derived_init_communicators
Dakota::DataFitSurrModel, 264
Dakota::NestedModel, 499
derived_initialize_run
Dakota::Iterator, 392
Dakota::LeastSq, 420
Dakota::Minimizer, 438
Dakota::NonD, 524
Dakota::Optimizer, 608
derived_map_ac
Dakota::DirectApplicInterface, 317
derived_master_overload
Dakota::NestedModel, 499
derived_parse_inputs
Dakota::NIDRProblemDescDB, 504
derived_post_run
Dakota::DDACEDesignCompExp, 309
Dakota::FSUDEesignCompExp, 336
Dakota::Iterator, 392
Dakota::LeastSq, 420
Dakota::Optimizer, 609
Dakota::ParamStudy, 647
Dakota::PSUADEDesignCompExp, 666
derived_sync
Dakota::SysCallApplicInterface, 754
derived_sync_nowait
Dakota::SysCallApplicInterface, 754
derived_synchronize
Dakota::DataFitSurrModel, 263
Dakota::HierarchSurrModel, 363
derived_synchronize_nowait
Dakota::DataFitSurrModel, 263
Dakota::HierarchSurrModel, 363
derived_synchronous_local_analysis
Dakota::DirectApplicInterface, 317
Dakota::ForkApplicInterface, 332
Dakota::GridApplicInterface, 356
Dakota::SysCallApplicInterface, 754
derivVarsVector
Dakota::ActiveSet, 157
Description
Dakota::JEGAOptimizer::Evaluator, 408
DestroyAlgorithm
Dakota::JEGAOptimizer::Driver, 405
DF
Dakota::CONMINOptimizer, 242
dg_ds_eval
  Dakota::NonDLocalReliability, 573
DiscSetLbl
  Dakota, 150
dl_api.C, 777
dakota_stop, 778
dl_api.h, 779
dakota_stop, 780
DoEval
  Dakota::COLINApplication, 219
dotFDInfo
  Dakota::DOTOptimizer, 320
dotInfo
  Dakota::DOTOptimizer, 320
dotMethod
  Dakota::DOTOptimizer, 321
Driver
  Dakota::JEGAOptimizer::Driver, 404
duplication_detect
  Dakota::ApplicationInterface, 174
ends
  Dakota::String, 723
enforce_input_rules
  Dakota::FSUDesignCompExp, 337
  Dakota::PSUADEDesignCompExp, 666
enroll
  Dakota::GetLongOpt, 349
ErrorTable, 328
estimate_derivatives
  Dakota::Model, 482
estimate_message_lengths
  Dakota::Model, 481
evalInterfaceIds
  Dakota::ParamResponsePair, 643
Evaluate
  Dakota::JEGAOptimizer::Evaluator, 410
evaluate_parameter_sets
  Dakota::Analyzer, 166
evaluation_id
  Dakota::DataFitSurrModel, 265
  Dakota::HierarchSurrModel, 363
  Dakota::NestedModel, 499
Evaluator
  Dakota::JEGAOptimizer::Evaluator, 407, 408
EvaluatorCreator
  Dakota::JEGAOptimizer::EvaluatorCreator, 412
EXP_TOO_BIG
  CtelRegexp, 256
expansionCoeffGrads
  Dakota::BasisPolyApproximation, 204
expectation
  Dakota::OrthogPolyApproximation, 619
  Dakota::SequentialHybridStrategy, 690
extract_parameter_sets
  Dakota::SequentialHybridStrategy, 690
extract_results_sets
  Dakota::SequentialHybridStrategy, 690
ExtractAllData
  Dakota::JEGAOptimizer::Driver, 404
factorial
  Dakota::BasisPolynomial, 210
factorial_ratio
  Dakota::BasisPolynomial, 211
fdGradStepSize
  Dakota::Iterator, 393
fdHessByFnStepSize
  Dakota::Iterator, 393
fdHessByGradStepSize
  Dakota::Iterator, 393
FDstep1
  Dakota::Model, 482
FDstep2
  Dakota::Model, 482
FIELD_NAMES
  Dakota, 89
finalize_run
  Dakota::Iterator, 391
find
  Dakota::List, 426
find_coefficients
  Dakota::SurfpackApproximation, 725
find_optimum
  Dakota::APPSOptimizer, 194
  Dakota::COLINOptimizer, 223
  Dakota::JEGAOptimizer, 401
flush
  Dakota, 86
  Dakota::CommandShell, 231
force_rebuild
  Dakota::SurrogateModel, 749
fork_application
  Dakota::ForkApplicInterface, 332
fpinit_ASL
  main.C, 785
free_iterator
  Dakota::Strategy, 720
FSUDesignCompExp
  Dakota::FSUDesignCompExp, 336
INDEX 831

functionGradients
  Dakota::ResponseRep, 684
functionSurfaces
  Dakota::ApproximationInterface, 188
G1
  Dakota::CONMINOptimizer, 241
G2
  Dakota::CONMINOptimizer, 241
gauss_points
  Dakota::BasisPolynomial, 209
gauss_weights
  Dakota::BasisPolynomial, 209
get
  Dakota::List, 425
get_approx
  Dakota::Approximation, 184
get_confidence_intervals
  Dakota::LeastSq, 421
get_constraints
  Dakota::Constraints, 254
get_db
  Dakota::ProblemDescDB, 660
get_gradient
  Dakota::BasisPolynomial, 209
  Dakota::LagrangeInterpPolynomial, 415
get_hessian
  Dakota::SurfpackApproximation, 726
get_interface
  Dakota::Interface, 374
get_iterator
  Dakota::Iterator, 393
get_mean
  Dakota::InterpPolyApproximation, 378
  Dakota::OrthogPolyApproximation, 617
get_mean_gradient
  Dakota::InterpPolyApproximation, 378, 379
  Dakota::OrthogPolyApproximation, 618
get_model
  Dakota::Model, 482
get_parameter_sets
  Dakota::NonDSampling, 589
get_polynomial
  Dakota::BasisPolynomial, 211
get_strategy
  Dakota::Strategy, 720
get_subsets
  Dakota::InterpPolyApproximation, 380
get_value
  Dakota::BasisPolynomial, 208

Dakota::LagrangeInterpPolynomial, 415
get_variables
  Dakota::Variables, 775
get_variance
  Dakota::InterpPolyApproximation, 379
  Dakota::OrthogPolyApproximation, 618
get_variance_gradient
  Dakota::InterpPolyApproximation, 379
  Dakota::OrthogPolyApproximation, 618, 619
GetBestMOSolution
  Dakota::JEGAOptimizer, 400
GetBestSolution
  Dakota::JEGAOptimizer, 400
GetBestSOSolution
  Dakota::JEGAOptimizer, 400
GetDescription
  Dakota::JEGAOptimizer::Evaluator, 411
getdist
  Dakota, 86
GetLongOpt
  Dakota::GetLongOpt, 349
GetName
  Dakota::JEGAOptimizer::Evaluator, 410
GetNumberLinearConstraints
  Dakota::JEGAOptimizer::Evaluator, 410
GetNumberNonLinearConstraints
  Dakota::JEGAOptimizer::Evaluator, 409
gerRmax
  Dakota, 86
GOOD
  CtelRegexp, 256
GPmodel_apply
  Dakota::GaussProcApproximation, 344
gradient_check
  Dakota::OrthogPolyApproximation, 620
hard_convergence_check
  Dakota::SurrBasedLocalMinimizer, 733
hom_constraint_eval
  Dakota::SurrBasedLocalMinimizer, 735
hom_objective_eval
  Dakota::SurrBasedLocalMinimizer, 734
IC
  Dakota::CONMINOptimizer, 242
id_vars_exact_compare
  Dakota, 87
increment_parallel_configuration
  Dakota::ParallelLibrary, 638
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dakota::List</td>
<td>426</td>
</tr>
<tr>
<td>INDEX_MATCH</td>
<td></td>
</tr>
<tr>
<td>CtelRegexp</td>
<td>257</td>
</tr>
<tr>
<td>INDEX_RANGE</td>
<td></td>
</tr>
<tr>
<td>CtelRegexp</td>
<td>256</td>
</tr>
<tr>
<td>init_communicators</td>
<td></td>
</tr>
<tr>
<td>Dakota::Model</td>
<td>481</td>
</tr>
<tr>
<td>Dakota::ParallelLibrary</td>
<td>638</td>
</tr>
<tr>
<td>init_iterator</td>
<td></td>
</tr>
<tr>
<td>Dakota::Strategy</td>
<td>719</td>
</tr>
<tr>
<td>init_iterator_parallelism</td>
<td></td>
</tr>
<tr>
<td>Dakota::Strategy</td>
<td>719</td>
</tr>
<tr>
<td>init_serial</td>
<td></td>
</tr>
<tr>
<td>Dakota::ApplicationInterface</td>
<td>173</td>
</tr>
<tr>
<td>Dakota::Model</td>
<td>481</td>
</tr>
<tr>
<td>initial_points</td>
<td></td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer</td>
<td>402</td>
</tr>
<tr>
<td>initial_taylor_series</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonDLocalReliability</td>
<td>572</td>
</tr>
<tr>
<td>initialize</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonDAdaptImpSampling</td>
<td>529</td>
</tr>
<tr>
<td>Dakota::NonDIntegration</td>
<td>550</td>
</tr>
<tr>
<td>Dakota::NonDQuadature</td>
<td>579</td>
</tr>
<tr>
<td>Dakota::NonDSparseGrid</td>
<td>594</td>
</tr>
<tr>
<td>Dakota::RecastModel</td>
<td>673</td>
</tr>
<tr>
<td>initialize_class_data</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonDLocalReliability</td>
<td>572</td>
</tr>
<tr>
<td>initialize_final_statistics</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonD</td>
<td>524</td>
</tr>
<tr>
<td>initialize_graphics</td>
<td></td>
</tr>
<tr>
<td>Dakota::Iterator</td>
<td>390</td>
</tr>
<tr>
<td>initialize_level_data</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonDLocalReliability</td>
<td>572</td>
</tr>
<tr>
<td>initialize_mpp_search_data</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonDLocalReliability</td>
<td>572</td>
</tr>
<tr>
<td>initialize_random_variable_parameters</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonD</td>
<td>525</td>
</tr>
<tr>
<td>initialize_random_variable_types</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonD</td>
<td>525</td>
</tr>
<tr>
<td>initialize_random_variables</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonD</td>
<td>524</td>
</tr>
<tr>
<td>initialize_run</td>
<td></td>
</tr>
<tr>
<td>Dakota::Iterator</td>
<td>390</td>
</tr>
<tr>
<td>initialize_scaling</td>
<td></td>
</tr>
<tr>
<td>Dakota::Minimizer</td>
<td>439</td>
</tr>
<tr>
<td>initialize_variables_and_constraints</td>
<td></td>
</tr>
<tr>
<td>Dakota::APPSOptimizer</td>
<td>194</td>
</tr>
<tr>
<td>insert</td>
<td></td>
</tr>
<tr>
<td>Dakota::List</td>
<td>426</td>
</tr>
<tr>
<td>INT_ERROR</td>
<td></td>
</tr>
<tr>
<td>CtelRegexp</td>
<td>257</td>
</tr>
<tr>
<td>intCntlParmArray</td>
<td></td>
</tr>
<tr>
<td>Dakota::DOTOptimizer</td>
<td>321</td>
</tr>
<tr>
<td>integration</td>
<td></td>
</tr>
<tr>
<td>Dakota::OrthogPolyApproximation</td>
<td>619</td>
</tr>
<tr>
<td>Interface</td>
<td></td>
</tr>
<tr>
<td>Dakota::Interface</td>
<td>373</td>
</tr>
<tr>
<td>interface</td>
<td></td>
</tr>
<tr>
<td>Dakota::Model</td>
<td>480</td>
</tr>
<tr>
<td>interface_id</td>
<td></td>
</tr>
<tr>
<td>Dakota::Model</td>
<td>480</td>
</tr>
<tr>
<td>interpolation_points</td>
<td></td>
</tr>
<tr>
<td>Dakota::BasisPolynomial</td>
<td>210</td>
</tr>
<tr>
<td>ISC</td>
<td></td>
</tr>
<tr>
<td>Dakota::CONMINOptimizer</td>
<td>242</td>
</tr>
<tr>
<td>isotropicSSG</td>
<td></td>
</tr>
<tr>
<td>Dakota::NonDSparseGrid</td>
<td>595</td>
</tr>
<tr>
<td>isWaiting</td>
<td></td>
</tr>
<tr>
<td>Dakota::APPSEvalMgr</td>
<td>191</td>
</tr>
<tr>
<td>Iterator</td>
<td></td>
</tr>
<tr>
<td>Dakota::Iterator</td>
<td>388</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td></td>
</tr>
<tr>
<td>Dakota::JEGAOptimizer</td>
<td>398</td>
</tr>
<tr>
<td>JEGAOptimizer.C</td>
<td>781</td>
</tr>
<tr>
<td>JEGAOptimizer.H</td>
<td>782</td>
</tr>
<tr>
<td>kw_1</td>
<td></td>
</tr>
<tr>
<td>Dakota, 90, 139</td>
<td></td>
</tr>
<tr>
<td>kw_10</td>
<td></td>
</tr>
<tr>
<td>Dakota, 92</td>
<td></td>
</tr>
<tr>
<td>kw_100</td>
<td></td>
</tr>
<tr>
<td>Dakota, 113, 143</td>
<td></td>
</tr>
<tr>
<td>kw_101</td>
<td></td>
</tr>
<tr>
<td>Dakota, 114</td>
<td></td>
</tr>
<tr>
<td>kw_102</td>
<td></td>
</tr>
<tr>
<td>Dakota, 114</td>
<td></td>
</tr>
<tr>
<td>kw_103</td>
<td></td>
</tr>
<tr>
<td>Dakota, 114</td>
<td></td>
</tr>
<tr>
<td>kw_104</td>
<td></td>
</tr>
<tr>
<td>Dakota, 114</td>
<td></td>
</tr>
<tr>
<td>kw_105</td>
<td></td>
</tr>
<tr>
<td>Dakota, 114</td>
<td></td>
</tr>
<tr>
<td>kw_106</td>
<td></td>
</tr>
<tr>
<td>Dakota, 115</td>
<td></td>
</tr>
<tr>
<td>kw_107</td>
<td></td>
</tr>
<tr>
<td>Dakota, 115, 144</td>
<td></td>
</tr>
<tr>
<td>kw_108</td>
<td></td>
</tr>
<tr>
<td>Dakota, 116</td>
<td></td>
</tr>
</tbody>
</table>
Dakota, 126
kw_154
Dakota, 127
kw_155
Dakota, 128
kw_156
Dakota, 127
kw_157
Dakota, 128
kw_158
Dakota, 127
kw_159
Dakota, 94
kw_16
Dakota, 128, 145
kw_160
Dakota, 128
kw_161
Dakota, 129
kw_162
Dakota, 128
kw_163
Dakota, 129
kw_164
Dakota, 128
kw_165
Dakota, 129
kw_166
Dakota, 129, 145
kw_167
Dakota, 94
kw_168
Dakota, 130
kw_169
Dakota, 130
kw_17
Dakota, 130
kw_170
Dakota, 94
kw_171
Dakota, 130
kw_172
Dakota, 130
kw_173
Dakota, 131, 145
kw_174
Dakota, 131
kw_175
Dakota, 131
kw_176
Dakota, 131
kw_177
Dakota, 132, 145
kw_178
Dakota, 132
kw_179
Dakota, 132
kw_18
Dakota, 94, 141
kw_180
Dakota, 133
kw_181
Dakota, 133
kw_182
Dakota, 133
kw_183
Dakota, 133
kw_184
Dakota, 134
kw_185
Dakota, 134
kw_186
Dakota, 134
kw_187
Dakota, 134
kw_188
Dakota, 135
kw_189
Dakota, 135
kw_19
Dakota, 19
kw_190
Dakota, 95
kw_191
Dakota, 135
kw_192
Dakota, 135
kw_193
Dakota, 136
kw_194
Dakota, 136, 145
kw_195
Dakota, 136, 146
kw_196
Dakota, 136
kw_197
Dakota, 137
kw_198
Dakota, 137, 146
| kw_199 | Dakota, 137 |
| kw_2   | Dakota, 90, 139 |
| kw_20  | Dakota, 95 |
| kw_200 | Dakota, 138 |
| kw_201 | Dakota, 138 |
| kw_202 | Dakota, 138 |
| kw_203 | Dakota, 138 |
| kw_204 | Dakota, 139 |
| kw_206 | Dakota, 139 |
| kw_207 | Dakota, 146 |
| kw_208 | Dakota, 147 |
| kw_209 | Dakota, 147 |
| kw_21  | Dakota, 95 |
| kw_210 | Dakota, 147 |
| kw_211 | Dakota, 147 |
| kw_212 | Dakota, 148 |
| kw_213 | Dakota, 148 |
| kw_214 | Dakota, 148 |
| kw_215 | Dakota, 148 |
| kw_217 | Dakota, 149 |
| kw_22  | Dakota, 95 |
| kw_23  | Dakota, 95 |
| kw_24  | Dakota, 96 |
| kw_25  | Dakota, 96 |
| kw_26  | Dakota, 96 |

Dakota, 96
kw_27  Dakota, 97
kw_28  Dakota, 97
kw_29  Dakota, 98
kw_3   Dakota, 98
kw_30  Dakota, 99
kw_31  Dakota, 99
kw_32  Dakota, 99
kw_33  Dakota, 99
kw_34  Dakota, 99
kw_35  Dakota, 99
kw_36  Dakota, 100
kw_37  Dakota, 100
kw_38  Dakota, 100
kw_39  Dakota, 100
kw_4   Dakota, 100
kw_40  Dakota, 101
kw_41  Dakota, 101
kw_42  Dakota, 101
kw_43  Dakota, 101
kw_44  Dakota, 101
kw_45  Dakota, 102
kw_46  Dakota, 102
kw_47  Dakota, 102
kw_48  Dakota, 102
kw_49
    Dakota, 102
kw_5
    Dakota, 90, 140
kw_50
    Dakota, 103
kw_51
    Dakota, 103
kw_52
    Dakota, 103
kw_53
    Dakota, 103
kw_54
    Dakota, 104
kw_55
    Dakota, 104
kw_56
    Dakota, 104
kw_57
    Dakota, 104
kw_58
    Dakota, 104, 142
kw_59
    Dakota, 105
kw_6
    Dakota, 91, 140
kw_60
    Dakota, 105, 142
kw_61
    Dakota, 105
kw_62
    Dakota, 105
kw_63
    Dakota, 105
kw_64
    Dakota, 106
kw_65
    Dakota, 106
kw_66
    Dakota, 106
kw_67
    Dakota, 106
kw_68
    Dakota, 106, 142
kw_69
    Dakota, 107
kw_7
    Dakota, 91, 141
kw_70
    Dakota, 107
kw_93
Dakota, 112
kw_94
Dakota, 112
kw_95
Dakota, 112, 143
kw_96
Dakota, 112
kw_97
Dakota, 113
kw_98
Dakota, 113
kw_99
Dakota, 113

lagrangian_merit
Dakota::SurrBasedMinimizer, 739

LeastSq
Dakota::LeastSq, 420
level_to_order_closed_exponential
Dakota::NonDSparseGrid, 594
level_to_order_open_exponential
Dakota::NonDSparseGrid, 594
level_to_order_open_linear
Dakota::NonDSparseGrid, 594
library_mode.C, 783
main, 784
model_interface_plugins, 784
my_callback_function, 784
run_dakota_data, 783
run_dakota_parsed, 783
run_dakota_parse, 783
lin_coeffs_modify_n2s
Dakota::Minimizer, 440

LoadAlgorithmConfig
Dakota::JEGAOptimizer, 399
LoadDakotaResponses
Dakota::JEGAOptimizer, 399
LoadProblemConfig
Dakota::JEGAOptimizer, 399
LoadTheConstraints
Dakota::JEGAOptimizer, 400
LoadTheDesignVariables
Dakota::JEGAOptimizer, 399
LoadTheObjectiveFunctions
Dakota::JEGAOptimizer, 400
LoadTheParameterDatabase
Dakota::JEGAOptimizer, 399

local_eval_concurrency
Dakota::Model, 480
local_eval_synchronization
Dakota::Model, 480
lookup_by_val
Dakota, 87, 88
lower
Dakota::String, 722
lower_sets
Dakota::InterpPolyApproximation, 380

main
library_mode.C, 784
main.C, 785
restart_util.C, 786
main.C, 785
fpinit_ASL, 785
main, 785
start_dakota_heartbeat, 785
manage_asv
Dakota::Model, 483
manage_inputs
Dakota::ProblemDescDB, 659, 660
manage_linear_constraints
Dakota::Constraints, 253
manage_outputs_restart
Dakota::ParallelLibrary, 638
MandatoryValue
Dakota::GetLongOpt, 349
map
Dakota::ApplicationInterface, 173
map_response
Dakota::COLINApplication, 219
merge_results_sets
Dakota::SequentialHybridStrategy, 690
MergedConstraints
Dakota::MergedConstraints, 429
MergedVariables
Dakota::MergedVariables, 431
mindist
Dakota, 86
mindistindx
Dakota, 86
minimize_surrogates
Dakota::SurrBasedLocalMinimizer, 733
Minimizer
Dakota::Minimizer, 438
MixedConstraints
Dakota::MixedConstraints, 442
MixedVariables
Dakota::MixedVariables, 444
Model
<table>
<thead>
<tr>
<th>Term</th>
<th>Class/Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dakota::Model</td>
<td>478</td>
</tr>
<tr>
<td>model_interface_plugins</td>
<td></td>
</tr>
<tr>
<td>library_mode.C</td>
<td>784</td>
</tr>
<tr>
<td>modify_n2s</td>
<td>Dakota::Minimizer, 439</td>
</tr>
<tr>
<td>modify_s2n</td>
<td>Dakota::Minimizer, 439</td>
</tr>
<tr>
<td>MS1</td>
<td>Dakota::CONMINOptimizer, 241</td>
</tr>
<tr>
<td>multi_objective_retrieve</td>
<td></td>
</tr>
<tr>
<td>Dakota::Optimizer,</td>
<td>609</td>
</tr>
<tr>
<td>my_callback_function</td>
<td>library_mode.C, 784</td>
</tr>
<tr>
<td>N1</td>
<td>Dakota::CONMINOptimizer, 240</td>
</tr>
<tr>
<td>N2</td>
<td>Dakota::CONMINOptimizer, 240</td>
</tr>
<tr>
<td>N3</td>
<td>Dakota::CONMINOptimizer, 240</td>
</tr>
<tr>
<td>N4</td>
<td>Dakota::CONMINOptimizer, 240</td>
</tr>
<tr>
<td>N5</td>
<td>Dakota::CONMINOptimizer, 240</td>
</tr>
<tr>
<td>n_choose_k</td>
<td>Dakota::BasisPolynomial, 211</td>
</tr>
<tr>
<td>Name</td>
<td>Dakota::JEGAOptimizer::Evaluator, 408</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>Dakota::NCSUOptimizer, 493</td>
</tr>
<tr>
<td>need_resp_trans_byvars</td>
<td></td>
</tr>
<tr>
<td>Dakota::Minimizer,</td>
<td>439</td>
</tr>
<tr>
<td>new_dataset</td>
<td>Dakota::Graphics, 353</td>
</tr>
<tr>
<td>next_eval</td>
<td>Dakota::COLINApplication, 219</td>
</tr>
<tr>
<td>nlf0_evaluator</td>
<td>Dakota::SNLLOptimizer, 709</td>
</tr>
<tr>
<td>nlf1_evaluator</td>
<td>Dakota::SNLLOptimizer, 710</td>
</tr>
<tr>
<td>nlf2_evaluator</td>
<td>Dakota::SNLLOptimizer, 710</td>
</tr>
<tr>
<td>nlf2_evaluator_gn</td>
<td>Dakota::SNLLOptSq, 703</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>Dakota::NLSSOLLeastSq, 516</td>
</tr>
<tr>
<td>NonDAdaptImpSampling</td>
<td>Dakota::NonDAdaptImpSampling, 528</td>
</tr>
<tr>
<td>NonDBayesCal</td>
<td>Dakota::NonDBayesCal, 531</td>
</tr>
<tr>
<td>NonDIncrementHSSampling</td>
<td>Dakota::NonDIncrementHSSampling, 547</td>
</tr>
<tr>
<td>NonDIntegration</td>
<td>Dakota::NonDIntegration, 549</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>Dakota::NonDLHSSampling, 559</td>
</tr>
<tr>
<td>NonDQuadrature</td>
<td>Dakota::NonDQuadrature, 579</td>
</tr>
<tr>
<td>NonDSampling</td>
<td>Dakota::NonDSampling, 588</td>
</tr>
<tr>
<td>NonDSparseGrid</td>
<td>Dakota::NonDSparseGrid, 594</td>
</tr>
<tr>
<td>norm_squared</td>
<td>Dakota::BasisPolynomial, 209</td>
</tr>
<tr>
<td>NPSOLOptimizer</td>
<td>Dakota::NPSOLOptimizer, 599</td>
</tr>
<tr>
<td>NUMBER_OF_FIELDS</td>
<td>Dakota, 89</td>
</tr>
<tr>
<td>objective</td>
<td>Dakota::SurrBasedMinimizer, 740</td>
</tr>
<tr>
<td>objective_gradient</td>
<td>Dakota::SurrBasedMinimizer, 740</td>
</tr>
<tr>
<td>operator const char *</td>
<td>Dakota::String, 722</td>
</tr>
<tr>
<td>operator T *</td>
<td>Dakota::Array, 197</td>
</tr>
<tr>
<td>operator&lt;&lt;</td>
<td>Dakota::BoStream, 217</td>
</tr>
<tr>
<td>operator=</td>
<td>Dakota::Approximation, 183</td>
</tr>
<tr>
<td>OPERATOR travellers</td>
<td>Dakota::Array, 197</td>
</tr>
<tr>
<td>optimizationType</td>
<td>Dakota::CONMINOptimizer, 239</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Dakota::Optimizer, 608</td>
</tr>
<tr>
<td>OptionalValue</td>
<td></td>
</tr>
<tr>
<td>RecastModel</td>
<td>Dakota::RecastModel, 673</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>RecordResponses</td>
<td>Dakota::JEGAOptimizer::Evaluator, 409</td>
</tr>
<tr>
<td>recv</td>
<td>Dakota::APPSEvalMgr, 191</td>
</tr>
<tr>
<td>regression</td>
<td>Dakota::OrthogPolyApproximation, 619</td>
</tr>
<tr>
<td>reliability</td>
<td>Dakota::NonDLocalReliability, 573</td>
</tr>
<tr>
<td>remove</td>
<td>Dakota::List, 425</td>
</tr>
<tr>
<td>removeAt</td>
<td>Dakota::List, 425</td>
</tr>
<tr>
<td>repair_restart</td>
<td>Dakota, 88</td>
</tr>
<tr>
<td>requestVector</td>
<td>Dakota::ActiveSet, 157</td>
</tr>
<tr>
<td>reset</td>
<td>Dakota::ResponseRep, 684</td>
</tr>
<tr>
<td>reset_gauss</td>
<td>Dakota::BasisPolynomial, 209</td>
</tr>
<tr>
<td>reset_inactive</td>
<td>Dakota::ResponseRep, 684</td>
</tr>
<tr>
<td>reshape</td>
<td>Dakota::Constraints, 253</td>
</tr>
<tr>
<td>sampling_reset</td>
<td>Dakota::NonDQuadrature, 579</td>
</tr>
<tr>
<td>SecondaryVariables</td>
<td>Dakota::JEGAOptimizer::Evaluator, 409</td>
</tr>
<tr>
<td>resolve_inputs</td>
<td>Dakota::ParallelLibrary, 638</td>
</tr>
<tr>
<td>resolve_samples_symbols</td>
<td>Dakota::NonDQuadrature, 579</td>
</tr>
<tr>
<td>Response</td>
<td>Dakota::Response, 678</td>
</tr>
<tr>
<td>response_mapping</td>
<td>Dakota::NestedModel, 499</td>
</tr>
<tr>
<td>response_modify_n2s</td>
<td>Dakota::Minimizer, 440</td>
</tr>
<tr>
<td>response_modify_s2n</td>
<td>Dakota::Minimizer, 440</td>
</tr>
<tr>
<td>ResponseRep</td>
<td>Dakota::ResponseRep, 681</td>
</tr>
<tr>
<td>restart_util.C, 786</td>
<td>main, 786</td>
</tr>
<tr>
<td>retrieve</td>
<td>Dakota::GetLongOpt, 349</td>
</tr>
<tr>
<td>returns_multiple_points</td>
<td>Dakota::JEGAOptimizer, 402</td>
</tr>
<tr>
<td>RIA_constraint_eval</td>
<td>Dakota::NonDReliability, 583</td>
</tr>
<tr>
<td>RIA_objective_eval</td>
<td>Dakota::NonDReliability, 583</td>
</tr>
<tr>
<td>RStatus</td>
<td>CtelRegexp, 256</td>
</tr>
<tr>
<td>run</td>
<td>Dakota::LeastSq, 420</td>
</tr>
<tr>
<td>run_dakota_data</td>
<td>Dakota::NonD, 524</td>
</tr>
<tr>
<td>run_dakota_mixed</td>
<td>Dakota::Optimizer, 608</td>
</tr>
<tr>
<td>run_dakota_parse</td>
<td>Dakota::PSStudyDACE, 662</td>
</tr>
<tr>
<td>run_iterator</td>
<td>Dakota::CONMINOptimizer, 241</td>
</tr>
<tr>
<td>run_sequential</td>
<td>Dakota::SequentialHybridStrategy, 690</td>
</tr>
<tr>
<td>run_sequential_adaptive</td>
<td>Dakota::SequentialHybridStrategy, 690</td>
</tr>
<tr>
<td>S</td>
<td>Dakota::CONMINOptimizer, 241</td>
</tr>
<tr>
<td>sampling_reset</td>
<td>Dakota::NonDQuadrature, 579</td>
</tr>
<tr>
<td>schedule_iterators</td>
<td>Dakota::Strategy, 720</td>
</tr>
<tr>
<td>secondaryResp_recast</td>
<td>Dakota::Minimizer, 439</td>
</tr>
<tr>
<td>self_schedule_analyses</td>
<td>Dakota::ApplicationInterface, 174</td>
</tr>
<tr>
<td>self_schedule_evaluations</td>
<td>Dakota::ApplicationInterface, 175</td>
</tr>
<tr>
<td>self_schedule_iterators</td>
<td>Dakota::Strategy, 720</td>
</tr>
<tr>
<td>SeparateVariables</td>
<td>Dakota::JEGAOptimizer::Evaluator, 409</td>
</tr>
<tr>
<td>serve_analyses_asynch</td>
<td>Dakota::JEGAOptimizer::Evaluator, 409</td>
</tr>
</tbody>
</table>
Dakota::ForkApplicInterface, 333
serve_analyses_synch
  Dakota::ApplicationInterface, 174
serve_evaluations
  Dakota::ApplicationInterface, 174
serve_evaluations_async
  Dakota::ApplicationInterface, 177
serve_evaluations_peer
  Dakota::ApplicationInterface, 177
serve_evaluations_synch
  Dakota::ApplicationInterface, 176
serve_iterators
  Dakota::Strategy, 720
set_apps_parameters
  Dakota::APPSOptimizer, 194
set_compare
  Dakota, 87
set_method_parameters
  Dakota::COLINOptimizer, 224
set_runtime_parameters
  Dakota::COLINOptimizer, 224
set_standard_method_parameters
  Dakota::COLINOptimizer, 223
set_u_to_x_mapping
  Dakota::NonD, 525
show_data_3d
  Dakota::Graphics, 353
SIM, 154
SIM::ParallelDirectApplicInterface, 623
SIM::SerialDirectApplicInterface, 692
smolyakMultiIndex
  Dakota::InterpPolyApproximation, 380
SNLLOptimizer
  Dakota::SNLLOptimizer, 709
solve_eigenproblem
  Dakota::InterpPolyApproximation, 380
solve_eigenproblem
  Dakota::InterpPolyApproximation, 380
tensor_product_terms
  Dakota::BasisPolyApproximation, 204
spawn
  Dakota::APPSEvalMgr, 191
spawn_analysis
  Dakota::SysCallAnalysisCode, 752
spawn_evaluation
  Dakota::SysCallAnalysisCode, 751
spawn_input_filter
  Dakota::SysCallAnalysisCode, 752
spawn_output_filter
  Dakota::SysCallAnalysisCode, 752
specify_outputs_restart
  Dakota::ParallelLibrary, 637
STARPLUS_EMPTY
  CtelRegexp, 256
STARPLUS_NESTED
  CtelRegexp, 256
STARPLUS_NOTHING
  CtelRegexp, 257
start_dakota_heartbeat
  main.C, 785
start_grid_computing
  Dakota, 87
static_schedule_evaluations
  Dakota::ApplicationInterface, 175
stop_evaluation_servers
  Dakota::ApplicationInterface, 174
stop_grid_computing
  Dakota, 87
Strategy
  Dakota::Strategy, 717, 718
subModel
  Dakota::NestedModel, 500
subordinate_iterator
  Dakota::Model, 479
subordinate_model
  Dakota::Model, 479
subordinate_models
  Dakota::Model, 481
surrogate_model
  Dakota::Model, 479
surrogates_to_surf_data
  Dakota::SurfpackApproximation, 726
synch
  Dakota::ApplicationInterface, 173
synch_nowait
  Dakota::ApplicationInterface, 173
synchronize
  Dakota::COLINApplication, 219
synchronize_derivatives
  Dakota::Model, 482
synchronous_local_analyses
  Dakota::ForkApplicInterface, 333
synchronous_local_evaluations
  Dakota::ApplicationInterface, 176
tensor_product_terms
  Dakota::BasisPolyApproximation, 204
ToDoubleMatrix
  Dakota::JEGAOptimizer, 401
TOO_MANY_PAR
  CtelRegexp, 256
total_order_terms
  Dakota::BasisPolyApproximation, 204
tr_ratio_check
  Dakota::SurrBasedLocalMinimizer, 734
TRAILING
  CtelRegexp, 257
truth_model
  Dakota::Model, 479
UNMATCH_PAR
  CtelRegexp, 256
unpack_parameters_buffer
  Dakota::ConcurrentStrategy, 233
  Dakota::SequentialHybridStrategy, 689
  Dakota::Strategy, 719
unpack_results_buffer
  Dakota::ConcurrentStrategy, 234
  Dakota::SequentialHybridStrategy, 689
  Dakota::Strategy, 719
update_actual_model
  Dakota::DataFitSurrModel, 265
update_approximation
  Dakota::ApproximationInterface, 187
  Dakota::DataFitSurrModel, 263, 264
update_augmented_lagrange_multipliers
  Dakota::SurrBasedMinimizer, 739
update_filter
  Dakota::SurrBasedMinimizer, 739
update_from_actual_model
  Dakota::DataFitSurrModel, 265
update_from_sub_model
  Dakota::RecastModel, 673
update_from_subordinate_model
  Dakota::Model, 480
update_lagrange_multipliers
  Dakota::SurrBasedMinimizer, 739
update_level_data
  Dakota::NonDLocalReliability, 572
update_mpp_search_data
  Dakota::NonDLocalReliability, 572
update_penalty
  Dakota::SurrBasedLocalMinimizer, 734
update_pma_reliability_level
  Dakota::NonDLocalReliability, 573
update_quasi_hessians
  Dakota::Model, 483
update_response
  Dakota::Model, 483
upper
  Dakota::String, 722
usage
  Dakota::GetLongOpt, 350
Valueless
  Dakota::GetLongOpt, 348
var_based_decomp
  Dakota::Analyzer, 166
var_mp_bgen
  Dakota, 151
var_mp_bgen_audi
  Dakota, 151
var_mp_bgen_audr
  Dakota, 151
var_mp_bgen_dis
  Dakota, 152
var_mp_bgen_eu
  Dakota, 152
var_mp_bndchk
  Dakota, 152
var_mp_ibndchk
  Dakota, 152
Variables
  Dakota::Variables, 773, 774
variables_recast
  Dakota::Minimizer, 439
vars_u_to_x_mapping
  Dakota::NonD, 525
Vlch
  Dakota, 150
VLS
  Dakota, 150
volumetric_quality
  Dakota::PStudyDACE, 662
weighted_sum
  Dakota::Optimizer, 609
write
  Dakota::ParamResponsePair, 642
  Dakota::ResponseRep, 682, 683
write_annotated
  Dakota::ResponseRep, 682
write_tabular
  Dakota::ResponseRep, 682